Attack Recognition Based on Kernel Attack Signatures *

A.W. Krings, W.S. Harrison, N. Hanebutte, C.A. Taylor
Computer Science Dept.
University of Idaho

M. McQueen
INEEL
Idaho Falls, Idaho

Abstract – This paper describes part of an ongoing project that specifies a new survivability architecture based on attack signatures. Specifically, it defines a process that attempts to generate accurate attack signatures at the lowest level, i.e. in the kernel, in a controlled operating system and network environment. These signatures are then used for two purposes, (1) to implement real-time attack recognition, and (2) to aid in a design process that uses methods of fault-tolerant systems design to achieve survivability. This research focuses on the first case.

Keywords — Attack recognition, system profiling, intrusion detection, kernel, attack signature.

1 Introduction

With the ever increasing number of networked computers on the Internet, the number of incidents involving malicious attacks on individual systems has become a major problem. The problems are ranging from benign activity, e.g. port sweeps, to Distributed Denial of Service (DDoS) attacks [2, 3]. The latter is often started from computers that have been overtaken by malicious hackers. It is crucial to recognize whether a computer has been compromised or is under attack. This has been the focus of much research in intrusion detection (ID) [11] and has resulted in a variety of intrusion detection systems (IDS), includes projects like EMERALD [9] or NetStat, a successor of NSTAT [8].

Abnormal behavior can be investigated at different levels, defining an IDS hierarchy, e.g. application, host, network or multi-network/infrastructure intrusion detection [2]. Two basic strategies are used to analyze both known and unknown attacks. The first is based on attack signature detection, the second considers anomaly detection. Ideally, attack recognition based on attack signatures requires a priori knowledge of *all* attack scenarios, whereas anomaly detection requires full knowledge of the expected behavior of the system to detect all attacks. Neither of these strategies can achieve complete realistic detection coverage [2].

The research presented here falls under the class of signature based approaches. Whereas in IDS detection is the main objective, the research presented here is a byproduct of a larger research picture. We are using attack signatures in a survivability architecture, in which the signatures serve as a mechanism to identify critical functionalities in an operating systems. These functionalities are then subjected to fault-tolerant hardening techniques. At the core of the survivability architecture are automatically generated low-level clean attack signatures. In this paper, issues of attack recognition of these signatures are our main focus.

Attack Signatures: Intrusion signatures can be specified as a sequence of events and conditions that lead to a break-in [7]. However, there is no consensus on this definition as signatures can contain different information including or excluding the notion of time or event sequencing. Many different interpretations exist, ranging from audit trail based signatures for use in expert systems [7, 10], to state oriented attack signatures [1]. Many projects or products use signatures, but, especially in commercial ID systems, due to the proprietary nature, it is not clear what the signatures constitute, e.g. *RealSecure* (Internet Security Systems) or *NetRanger* (WheelGroup Inc.). In general, the lack of accessibility to signatures leaves a general desire to support the growing interest in open-source signatures [2].

Most signatures are derived from audit records that record system events. When signatures contain specific sequences of system events involved in malicious actions, one often speaks of traces. It is important to realize at which level of complexity signatures are formed. Most signatures have been based on discrete events that are extracted from system log files or network packet sequences. These data are however at a relatively high level of abstraction. At a lower level of complexity one can derive signatures of system calls. Such approaches have been investigated in [6, 12]. However, at the 3^{rd} Information Survivability Workshop (ISW-2000), the key note speaker Peter Neumann (SRI) pointed out the importance of low level approaches to attack recognition. Kernel based

*This work has been supported in part by an LDRD grant of the Idaho National Engineering and Environmental Laboratory (INEEL).

intrusion detection is positioned at the lowest level of observation, just above hardware assisted detection. The benefits of kernel based intrusion detection have only been discussed in the literature in the recent past. Some kernel based intrusion detection systems are based on wrappers [4], or on dynamic profiling [5]. Our research is based on kernel profiling.

1.0.1 Back to Basic Measuring

The focus of this research is to turn the attack recognition problem into a basic dynamic measuring problem. As stated above, this work does not claim to present comprehensive answers to general intrusion detection. It focuses on specific attack recognition that will be a component of a survivability architecture currently in development. Since we assume that some survivability features need to be employed at the kernel level, we also need to be able to detect attacks targeting critical functionalities at that level.

At the heart of our approach to attack recognition is the process of clean measurement. We want to measure the impact of specific attack scenarios on the system as accurately as possible. Once the measurements are taken and "cleaned up" they can constitute a frame of reference. As in any IDS, in order to identify when a system behavior is "abnormal", it is important to establish what the "normal" system should look like. In [5] a normal system was considered as the activities of the kernel under four distinct application environments. The baseline profile was then established from these exercises. Attack recognition was based on the detection of (1) decreased activities of baseline profiles in conjunction with (2) increased activities of baseline profiles caused by an attack.

We can demonstrate that one can achieve attack recognition, independent of application environments, based on the quality and analysis of the attack signatures alone. The baseline of our investigations is the measurement of an idle system in its purest form.

We model our measurement procedure on measurements of physical phenomena. First, we focus our attention on the network portion of the target system, a linux based host. Next, in order to reduce system noise, we conduct all measurements off-line, eliminating any unrelated network activity. Specifically, we look at two connected linux based systems in isolation. One system serves as an attacker, the other as the attack target. Lastly, there are no applications executing, i.e. the systems are running as console (without xwindows support). Now, signatures were extracted from an absolutely idle system as it was subjected to very narrowly defined attacks, as will be described in Subsection 2.2.

2 Model Definition

2.1 The Target System

Our research focus is to apply very specific attack recognition at the lowest level. As such, we implement host based attack recognition. Thus, our target is a single networked computer operating in a *standard user environment*. We view such an environment as a typical desktop computer, operated mostly by single individuals. Given the affordability of desktop computers and the increasing popularity of linux, it is our experience in the academic environment that most students have fairly powerful desktop computers as their standard networked working console in private settings or university labs. This is quite different from recent environments where perhaps small numbers of users connected to the same unix host via inexpensive xterminals. The usage of standard, "dedicated", work stations is in general very low. Most common user profiles include applications such as x-windows, browsers, email, compilers, or small web servers. However, unlike systems such as transaction systems or main servers, the actual utilization are generally surprisingly low, as can be verified on individual systems using utilities such as `top`.

The basic system model can be seen in Figure 1. The target environment is a medium desktop computer, e.g. a Pentium III, running RedHat linux 6.2 kernel version 2.2.16. Such a system consists of the general operating system, including file system or memory management, as well as some functionalities we may consider critical, e.g. accesses to password files or shared files. Our interest in attack signatures is mainly focused on the network interface of the operating system, which is indicated as a separate block. The main protocol stack is TCP/IPv4. Next, we want to introduce our formal model.

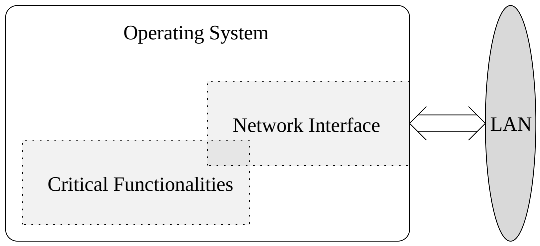


Figure 1: System Architecture

2.2 Profiles and Signatures

Similarly to [5], we view the system as a collection of functionalities. The functionalities are observed during specified time intervals Δt. Specifically, we view a system in terms of its system profile $P_{sys}(\Delta t)$, which is composed of the profiles of all functionalities $P_i(\Delta t)$ executing during Δt. Thus, for any Δt we have

$$P_{sys}(\Delta t) = \sum_{i=1}^{k} P_i(\Delta t)$$

where k is the number of functionalities active during the time interval. Each $P_i(\Delta t)$ is a vector of a length equal to the number of identities F, i.e. C functions, profiled. Therefore, if there are n identities profiled, then

$$P_i(\Delta t) = (f_1(\Delta t), f_2(\Delta t), ..., f_n(\Delta t)),$$

where $f_j(\Delta t)$, $1 \leq j \leq n$, is the number of times function F_j has been called (or activated) during Δt. A value of $f_j(\Delta t) = 0$ implies that function F_j has not been invoked at all, whereas $f_j(\Delta t) = x$, x a positive integer, implies that F_j has been invoked x times during Δt.

Simple Attack Signatures: Attacks are malicious attempts to gain access to restricted resources or to gather information on a system in order to expose vulnerabilities. In practice, attacks may be mounted using individual programs or attack suites, e.g. *toast* (http://breedersec.virtualave.net/code.html). We assume attacks to be atomic. An atomic attack A_i is the smallest attack technology unit[1], e.g. a port sweep or a sequence of unsuccessful login attempts. Thus, an attack suite can be viewed as a collection of individual atomic attacks A_i. Limiting the scope of an attack to atomic units allows us to focus on a very narrow sets of affected functions in the OS, application and network.

Profiles during atomic attacks are of special interest and result in attack signatures, i.e. an attack signature is the portion of a profile that is attributable to the attack. Formally, the attack signature corresponding to A_i will be denoted by S_i. Only non-zero profile components are considered. Thus,

$$S_i = (f_{\alpha(1)}(\Delta t), f_{\alpha(2)}(\Delta t), ..., f_{\alpha(s_i)}(\Delta t)),$$

where α is a function that maps (one-to-one) the indices of S_i to the indices of the functions profiled. Note that s_i, the length of vector S_i, is signature dependent.

A signature of special interest is the signature of an idle system. The so-called *idle signature* is denoted by S_0. Signature S_0 corresponds to the system profile $P_{sys}(\Delta t)$ of an idle linux system. Contrary to [5] our idle profile constitutes the system profile of a linux system that was just booted up. No applications, e.g. x-windows, are executing.

[1] We are not addressing issues of running attacks in stealth mode.

2.3 Ideal Signature Analysis

Once a signature S_i has been extracted, the question arises how this signature will be recognized in the noisy profile of a running system, i.e. $P_{sys}(\Delta t)$. We limit our investigation on the effectiveness of the simplest analysis in order to minimize run-time overhead.

First, we assume an ideal scenario in which attack signatures are absolutely accurate, i.e. there is no system noise, and the time interval Δt precisely captures the entire attack. Obviously, such ideal scenario will be unachievable, however, it will serve as a frame of reference in the realistic environment discussed in Subsection 2.4.

Let $\mathbf{S}_i$ be the set of functions F_j that are part of signature S_i, i.e. $\mathbf{S}_i = \{F_{\alpha(j)} \text{ in } S_i, 1 \leq j \leq s_i\}$. By the definition of a signature we have $f_{\alpha(j)} > 0$. Furthermore, let $|\mathbf{S}_i|$ denote the cardinality of $\mathbf{S}_i$. Let $\mathbf{P}_{sys}(\Delta t)$ be the set of functions F_j with non-zero frequencies, i.e. $f_j(\Delta t) > 0$, in the system profile. Thus, the number of functions represented in $\mathbf{P}_{sys}(\Delta t)$ is less than or equal to the number of functions in $P_{sys}(\Delta t)$, i.e. functions not used in Δt are excluded. Similar to $|\mathbf{S}_i|$, let $|\mathbf{P}_{sys}(\Delta t)|$ denote the cardinality of the run-time system profile. Now the following lemma can be stated:

Lemma 1 *Given an ideal run-time profile $P_{sys}(\Delta t)$ and a signature S_i, if $\mathbf{P}_{sys}(\Delta t) \cap \mathbf{S}_i \neq \mathbf{S}_i$ then attack A_i cannot be present.*

Proof: By the very definition of signature S_i, the absence of any $f_{\alpha(j)}$, $1 \leq j \leq s_i$, invalidates S_i. $\square$

Lemma 1 does not consider the frequencies $f_{\alpha(j)}$ as it only considers sets. However, signatures are characterized by the implied frequencies of the functions in $\mathbf{S}_i$.

Profiles and signatures are vectors. As such, relations between them have to be considered in the context of vector operations. The difference in length of $P_{sys}(\Delta t)$ and S_i complicate matters of comparison. Therefore we introduce a function $\mathcal{T}_i(X)$, that translates a profile X into a profile of length s_i such that only the functions in S_i are reflected. Thus, $P = \mathcal{T}_i(P_{sys}(\Delta t))$ generates a profile P that is of the same length as S_i, whose elements reflect the frequencies of the same functions, i.e. the functions of S_i.

We use 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.

Before stating the lemma that considers the relationship of profiles and signatures we want to define two terms. A *false positive* is an event incorrectly identified by the IDS as an attack when none has occurred. Similarly, in the presence of an attack, failure to recognize the attack constitutes a *false negative* [2].

Lemma 2 *Given an ideal run-time profile $P_{sys}(\Delta t)$ and a signature S_i, let $P = \mathcal{T}_i(P_{sys}(\Delta t))$. Then*

$$\begin{array}{llll} a) & \text{if} & P \geq S_i & \text{then } A_i \text{ possible} \\ b) & \text{if} & P \not\geq S_i & \text{then } A_i \text{ not possible} \\ c) & \text{if} & P < S_i & \text{then } A_i \text{ not possible} \end{array}$$

Proof: In case a) P and thus $P_{sys}(\Delta t)$ dominate S_i. The functionalities contributing to P cover or drown S_i. $P_{sys}(\Delta t)$ could include A_i, in which case S_i would be embedded in P. In the absence of A_i this scenario constitutes a false positive.

Case b) is of special interest, since there must exist some index p and q such that $f_p^P < f_p^{S_i}$ and $f_q^P > f_q^{S_i}$. The superscripts indicate which vector frequency f belongs to, e.g. to P or S_i. At index q, the function frequency of f_q is insufficient to include S_i. If $f_q^P = 0$ then A_i is not possible by Lemma 1.

In case c) S_i has higher frequencies than P for every function, i.e. $f_j^{S_i} > f_j^P$ for all F_j, $1 \leq j \leq s_i$. Thus function frequencies are insufficient to include S_i. □

2.4 Realistic Signature Analysis

The assumptions in Subsection 2.3 may be unrealistic. The quality of the attack signature depends on the quality of the process that generates S_i. Attack signatures are generated in the environment described previously by repeating the measuring process numerous times and applying filtering methods. The signature can be verified against the call graph of the system.

The second assumption about a perfect Δt during run-time profiling is unrealistic. Implications of Δt on $P_{sys}(\Delta t)$ need to be considered. One could set Δt to the duration determined during the measurement of S_i, and use a sliding window approach to determine $P_{sys}(\Delta t)$ in order to avoid the attack falling between two time intervals. This approach works under low system utilization. However, it is possible that system resources become saturated, e.g. the load of the system is high, resulting in functionalities to take longer than under low system load. Such behavior has direct implications to cases b) and c) of Lemma 2. The effect is that Δt needs to be adjusted to compensate for the slowdown. Such adjustments can be made under consideration of system load indicators, e.g. *uptime*.

In a realistic scenario that allows resource saturation one has to consider the implications of selecting an appropriate Δt. Define an array of random variables X_i, $1 \leq i \leq s_i$, that represents the invocations of F_i caused by A_i. Denote probabilities of event e with $Pr[e]$. Given a run-time profile $P_{sys}(\Delta t)$, signature S_i, and $P = \mathcal{T}_i(P_{sys}(\Delta t))$, then

a) if $P \geq S_i$, and no A_i has occurred, then a false positive has probability $\prod_{j=1}^{s_i} Pr[X_j \geq f_j^{S_i}]$,

b) if $P \not\geq S_i$ and A_i has occurred, then probability of a false negative is $\prod_{j=1}^{s_i} Pr[X_j < f_j^{S_i}]$, (note, that no functions with $f_j^P \geq f_j^{S_i}$ contribute to the product),

c) if $P < S_i$ and A_i has occurred, then probability of a false negative is $\prod_{j=1}^{s_i} Pr[X_j < f_j^{S_i}]$. Note that *all* functions contribute to the product.

3 Experimental Results

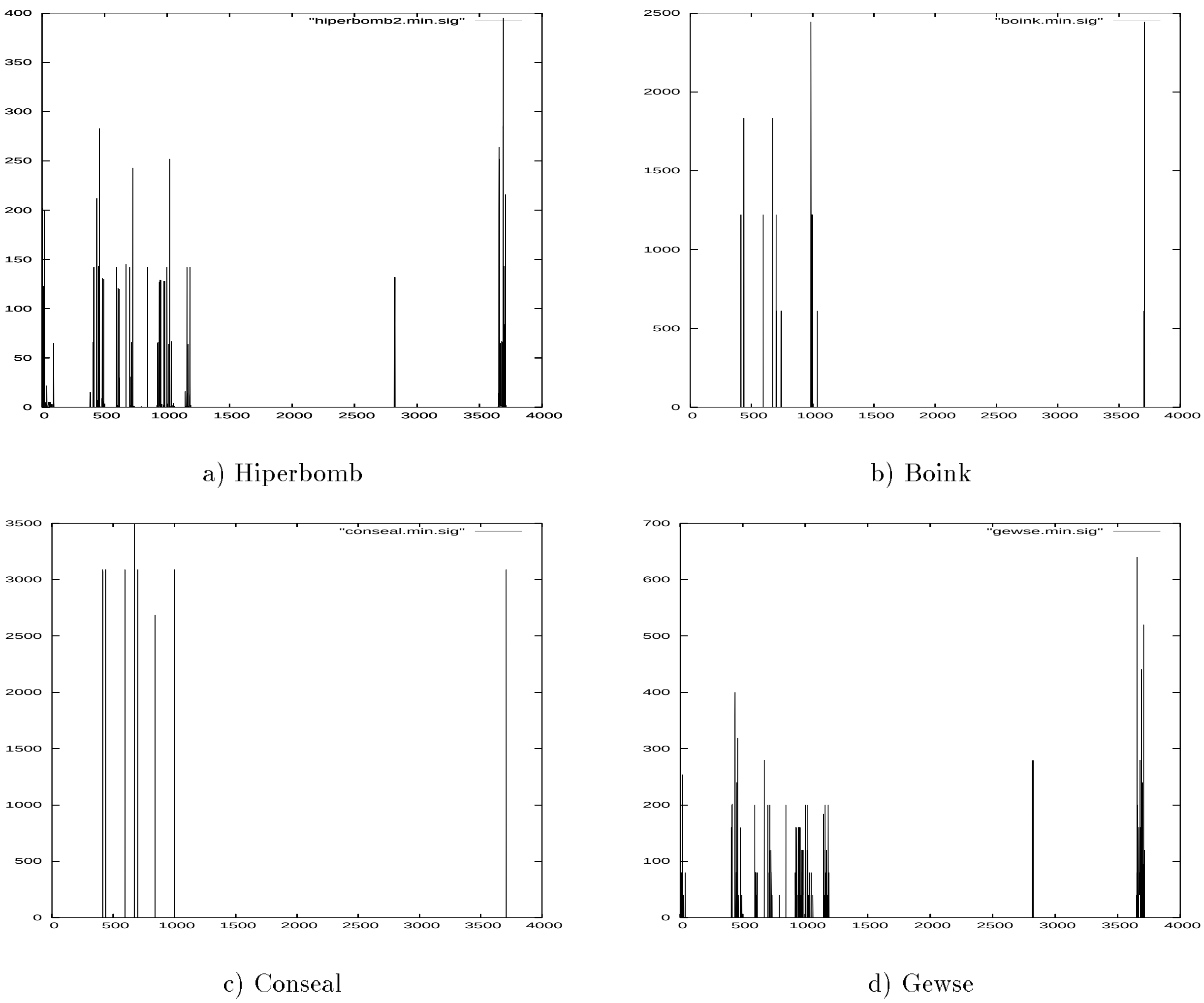
3.1 Signature Libraries

To collect signatures of attacks in a laboratory setting, one first must select an appropriate Δt. As discussed in Section 2.4, this is, in a real-world situation, not a simple task. Observation of an unloaded, idle system (defined below) shows that the majority of attacks on the network, e.g. "misfrag" or "conseal", take place within a few milliseconds. To ensure the entire attack is captured, we experimentally set Δt to one second.

Once this was done, the collection machine was rebooted and in an isolated network, the attack was run against this machine several times, as many as 50 times, with several seconds pause between attacks. The data was collected, the machine was rebooted, and the next attack was run.

After all the data for a particular attack was collected, a 3-dimensional plot of the profiled network activity was generated. In general, the attacks are very distinctive "mountain ranges" in an otherwise zero-valued dataset. These datasets were examined by hand to determine the good examples of the attack that exist in the data. Excluded are cases when the machine does not capture the entire attack.

Figure 2 illustrates the signatures collected for four different attacks, "hiperbomb," "boink," "conseal," and "gewse". The X-axis is the function index corresponding to our instrumentation. For example, function number 2693 corresponds to the `packet_rcv_spkt` function inside the `af_packet.c` file.



a) Hiperbomb b) Boink

c) Conseal d) Gewse

Figure 2: Examples of Attack Signatures

Currently there are 3818 such instrumented functions. The Y-axis corresponds to the frequency of function calls for the experimental Δt, which is set to one second. Note that $\Delta t = 1s$ has only been determined useful experimental, however, Δt may be adjusted dynamically.

3.2 Correlation between Signatures

Signatures S_i indicate the frequencies of the functions involved. In order to see how related the signatures are, with respect to the functions they share, we investigated their correlation. Given two signatures S_i and S_j, $1 \leq i, j \leq a$, where a is the total number of attacks considered, we define correlation $C(i, j)$ as

$$C(i,j) = \frac{|\mathbf{S}_i \cap \mathbf{S}_j|}{min(|\mathbf{S}_i|, |\mathbf{S}_j|)}.$$

Figure 3 illustrates the correlations between a sample set of signatures. The attacks A_i are listed from left-to-right and top-to-bottom, sorted by their size $|\mathbf{S}_i|$. Increasing level of darkness indicates stronger correlation. Again, only correlation of functions, *not* frequencies, is shown. Squares are color coded as follows: black squares represent values of $C(i, j) = 1.0$, e.g. $C(1234, bonk) = 1.0$, indicating that $\mathbf{S}_{1234} \subseteq \mathbf{S}_{bonk}$, and thus all the functions in "bonk" are also called in "1234". The dark grey squares represent values between $C(i, j) = [0.75, 1.00)$, light gray squares represent values from $C(i, j) = [0.5, 0.75)$, and white squares represent values from $C(i, j) = [0.0, 0.5)$.

3.3 The Attack Detection System

The Attack Detection System used at runtime needs to be efficient. In order to avoid processing the entire lengthy profile $P_{sys}(\Delta t)$ in the search for attack

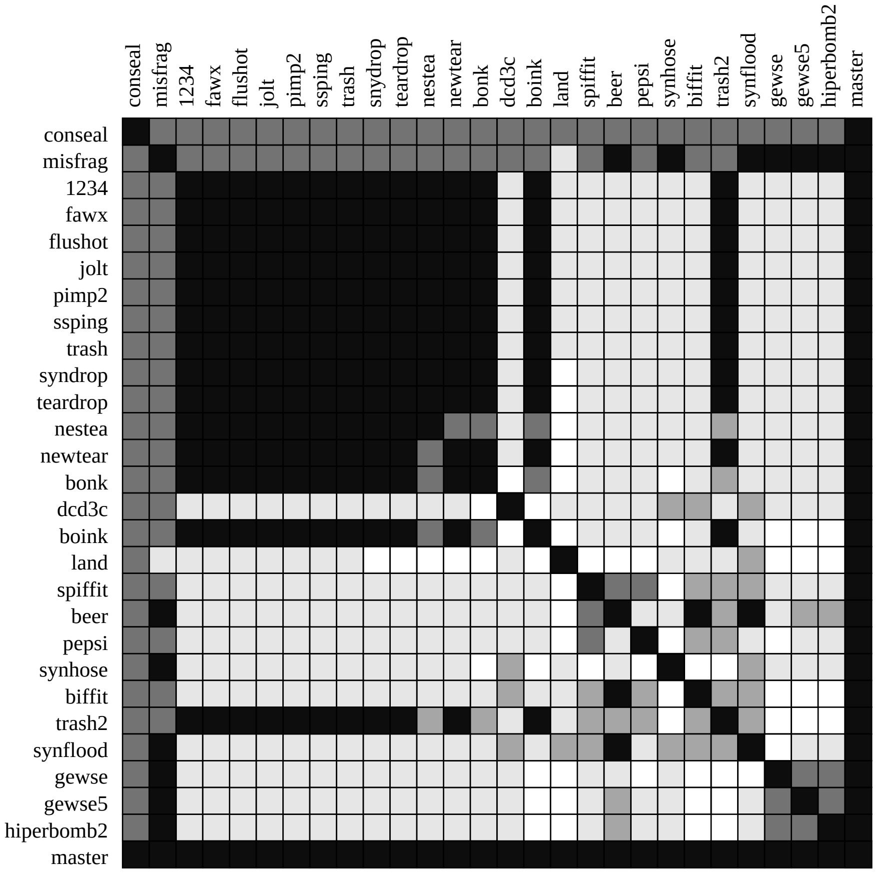


Figure 3: Signature Correlations

signatures S_i, we define set

$$\mathbf{S} = \bigcup_i \mathbf{S}_i.$$

This reduces overhead by focusing comparisons of $P_{sys}(\Delta t)$ to only include functions in $\mathbf{S}$, e.g. in our experiments, the maximum value for $|\mathbf{P}_{sys}(\Delta t)|$ is bound to 3818, and $|\mathbf{S}| = 320$. During runtime only the frequencies of functions in $\mathbf{P}_{sys}(\Delta t)$ which correspond to functions within $\mathbf{S}$ are compared. Classification of attacks is made according to the three scenarios in Section 2.4.

Idle System: In order to ensure that the signature we collect is clean and not the result of other activity on the target machine, we must ensure that the "idle profile" shows little or no activity.

In the lab, we were able to verify experimentally that the idle profile is, in fact, surprisingly idle. Specifically, if we record activity over several hours with an idle machine, we get virtually no activity. Note that we are profiling only the networking portion of the operating system. Occasionally, we do see functions called, but these have a very low magnitude, typically under 10 calls to a particular function per hour. In the derivation of a signature we collect the results from many instances of the same attack and take only the common functions in *all* attack instances. This effectively filters out idle noise. The probability of generating a noisy signature S_i thus reduces to the product of the probabilities that the same noise occurs in *each* instance of the attack. This probability is further reduced, due to the small value of the experimental Δt.

3.4 False Positives/Negatives

Only extensive field testing will reveal the actual percentage of false positives. As stated in [2], it is very difficult, if not impossible to give accurate numbers. At this stage of the project we have not succeeded in generating pathological scenarios for non-trivial attacks that result in false positives. Our testing assumed running the Attack Detection System under nominal system usage, including applications such as x-windows, telnet sessions, ftp sessions, and web page retrieval to and from the target machine. However, we have reported false positives for attacks with very small numbers of functions, e.g. "misfrag", which were the result of benign activity.

With respect to false negatives, our automatic attack scripts have revealed that less than 1% of the attacks were not recognized. However, more elaborate testing procedures are underway.

On the other hand, if the frequencies of the functions within highly correlated signatures are similar, several of the respective attacks may be recognized. This is mainly due to the high correlation between signatures, as was shown in Figure 3.

Overhead of Instrumentation: The overhead of this instrumentation is low, and limited to simple entry- and exit-point manipulations of the array containing the profiling information used to determine $P_{sys}(\Delta t)$. This array uses approximately 15KBytes of memory. Each signature uses approximately 2KBytes.

The runtime overhead consists of checking and is relative to the size of $\mathbf{P}_{sys}(\Delta t) \cap \mathbf{S}$. Although we only process $P_{sys}(\Delta t)$ every second, we can analyze approximately 30 signatures in 25 μs milliseconds on an unloaded Pentium 3 at 800 MHz. With linear scaling, this would indicate that 300 signatures could be analyzed in $250\mu s$, and even 3000 signatures in 2.5 ms, for a total overhead of 2.5%.

4 Conclusions

This research presented a very low overhead, kernel based, approach to attack recognition. Specifically, attack signatures were derived in a controlled operating system and network environment. These signatures build the basis for a real-time attack recognition system. Whereas our signature based method itself may not provide comprehensive intrusion detection, it recognizes specific attacks at the lowest level, both in terms of complexity and overhead. The method is therefore especially suited for a layered approach to attack recognition, taking advantage of low overhead limited recognition at the kernel level, while increasing recognition capabilities using slower, but more comprehensive, high complexity intrusion detection systems.

References

[1] Sobirey, Richter, and Konig, The intrusion detection system AID. Architecture, and Experiences in Automated audit Analysis, *Proceedings of the IFIP TC6/TC11 International Conference on Communications and Multimedia Security*, pp. 278-290, September 1996.

[2] Allen, Julia et.al., State of the Practice of Intrusion Detection Technologies, Carnegie Mellon, SEI, Technical Report, CMU/SEI-99-TR-028, ESC-99-028, January 2000.

[3] CERT Coordination Center, Carnegie Mellon, SEI, http://www.cert.org/advisories.

[4] Ko, Calvin, Timothy Fraser, Lee Badger, and Douglas Kilpatrick, Detecting and Countering System Intrusions Using Software Wrappers, *Proceedings of the 9^{th} USENIX Security Symposium*, 2000.

[5] Elbaum, Sebastian, and John Munson, Intrusion Detection Through Dynamic Software Measurement, *Proceedings of the Eighth USENIX Security Symposium*, 1999.

[6] Steven A. Hofmeyr, and Stephanie Forrest, Intrusion Detection using Sequences of System Calls, *Journal of Computer Security*, Vol. 6, pp. 151-180, 1998.

[7] Sandeep Kumar, and Eugen H. Spaffort, An Application of Pattern Matching in Intrusion Detection, Technical Report CSD-TR-94-013, The COAST Project, Purdue University, 1994.

[8] Kemmerer, Richard, NSTAT: A Model-Based Real-Time Network Intrusion Detection System (RTCS97-18), November 1997, http://www.cs.ucsb.edu/~kemm/netstat.html/-documents.html.

[9] Neumann, Peter, and Phillip Porras, Experience with EMERALD to DATE, *Proc. 1st USENIX Workshop on Intrusion Detection and Network Monitoring*, Santa Clara, California, pp. 73-80, 1999.

[10] Porras Phillip, and Peter G. Neumann, EMERALD: Event Monitoring Enabling Response to Anomalous Live Disturbances, *1997 National Information Systems Security Conference*, 1997, http://www.sdl.sri.com/emerald/emerald-niss97.html.

[11] Purdue University, Coast intrusion detection, http://www.cerias.purdue.edu/coast/intrusion-detection/ids.html.

[12] C. Warrender, S. Forrest, B. Pearlmutter, Detecting Intrusions Using System Calls: Alternative Data Models, *1999 IEEE Symposium on Security and Privacy* , pp. 133-145, 1999.