

Spyros Ligouras - 563

Paper report for: 'Automated Response Using System-Call Delays', by Somayaji, A., and S. Forrest, published in Usenix Security Symposium 2000

Introduction

In this paper the authors introduce a mechanism that can automatically respond to attacks or anomalies in general, in order to maintain the system's integrity. Its core technique is the introduction of time delays between system-calls of a running process.

Motivation

At the time of writing the paper, there had been little research in automated response systems, as mostly security research had mostly focused on Intrusion Detection & Prevention. Unfortunately, the problem with IDSs is that they need constant management in order to minimize false positives. On the other hand, a system that can respond automatically in certain threats and anomalies is more than welcome.

The major contributions of the paper are the following:

- i) The authors showed that monitoring every process at system-call level in real-time is feasible.
- ii) They introduced a practical method automatically responding to abnormal program behavior.

Anomaly Detection Systems rely on normal usage or network traffic profiles. These profiles are created by repeating and logging successful normal executions of a program so that the system will learn what constitutes a legitimate execution (training). What they do is monitor the processes or the network traffic for anomalies according to those profiles and alert the system administrator when they detect something abnormal.

In this case, it is almost impossible to eliminate “false positives”, mainly for three reasons:

- i) These systems are usually deployed in completely dynamic environments where it is possible to encounter cases of legitimate execution that the system doesn't know of. That is called “perpetual novelty”.
- ii) The profiles of normal usage are constantly changing as new hosts or users are added to the system.
- iii) There is an inherent ambiguity in the distinction between normal and abnormal activities e.g. changes to privileged files are legitimate for a system administrator but not for any other user.

pH

pH (process homeostasis) is an automated response system that allows a computer to preserve its integrity. What it does is to add a mechanism delaying or aborting an illegal system-call execution. It also provides the means for logging all system-calls executed by a process. By using pH we can “teach” the system that the behavior of a process is legit or to insert new data of a legitimate behavior for a specific process.

The methodology used is the following: monitor every process on a system-call level. For each new process record a time series of all the system-calls of the process to a trace. The goal was to be

able to extract from the trace the appropriate data that can show when an execution is legitimate and create an empirical model of its normal behavior.

This model must be:

- i) suitable for on-line training and testing
- ii) suitable for a large alphabet sizes, meaning system-calls (i.e. Unix has ~200 different system-calls).
- iii) Sensitive to common forms of intrusion.

The authors chose to use the “Time - delay Embedding” model. In the model normal behavior is defined as a series of system-calls. A sliding window is used on a trace and pairs of system-calls are recorded (current-previous system-call) in the current window. The sum of all these unique pairs, of all “traces” is the legitimate usage model for the specific program..

While testing, live system-calls pairs are compared to the ones specified in the model and if a certain pair does not exist in the model, then we have a mismatch. This mismatch can be either true positive or false positive and the system-call that caused it is considered anomalous.

Anomalies caused by attacks usually appear in (time) clusters so in order to get better results with identifying attacks, a circular array is used that holds all the recent anomalous system-calls. Each array element has the value 1 if the equivalent system-call caused 1 or more mismatches. Finally, the total number of anomalies is noted as $LFC = \sum A_i$.

pH mainly does two things:

- i) It monitors all processes in a system-call level
- ii) It responds automatically to any threats either by delaying anomalous system calls or by aborting execution.

pH was implemented in kernel space as other approaches such as audit packages, user-space tracing utilities and instrumented libraries either added to much overhead couldn't record properly all system-calls. Also implementation in kernel space minimized I/O cost while maximizing efficiency, stability and security.

pH maintains two arrays of data. First, the training array that is continuously updated with the system-calls as they appear and the testing array that contains the current legitimate execution profile of the process.

Replacement of the testing array occurs under three circumstances:

- i) By the user, via a special system call (sys_pH).
- ii) When the profile anomaly count exceeds the parameter anomaly_limit, and that is done so that the system can “learn” new behaviors.
- iii) When the training formula is satisfied.

When a system-call is identified as anomalous then it is met with a delay of $d \times \text{delay_factor}$ where $d=2^{LFC}$ and delay_factor is a user-specified parameter. Another user-specified parameter, abort_execve defines the the anomaly threshold which prevents the process from spawning new processes when net. For each program the maximum value of LFC is saved into maxLFC and if $\text{maxLFC} > \text{abort_execve}$ all excve calls are discarded. Finally, if $LFC > \text{tolerization_limit}$ (=12) the training array resets to avoid replacing the testing array with faulty data.

According to pH's design, programme profiles are stored in the disk and are loaded when a new process spawns. Each process' profile contains both the training and testing arrays.

Experiments

In the experiments, three known vulnerabilities where used:

- An SSH daemon backdoor

- An SSH daemon buffer-overflow
- A sendmail attack that exploits a bug into linux kernel's capabilities code

Exploits of the above always result in the attacker gaining root access.

In the case of the sshd backdoor, we assume that `delay_factor=4`. PH detects the attack and introduces a delay that increases exponentially until finally a shell is spawned. Setting `abort_execve=1` disables the attack, closing successfully the backdoor.

The buffer-overflow attack on sshd caused 4 ανωμαλίες. By setting `delay_factor=4` the same number of anomalies was detected but an 80 second delay was introduced but the attacker could still exploit the vulnerability. Enabling `abort_execve` managed to thwart the attack.

The attack on sendmail caused many anomalies, reaching the `tolerization_limit` several times. Enabling `abort_execve` didn't work in this case, but using `delay_factor=4` managed to deal with the attack, as it introduced a delay of about two hours.

The overhead of using pH and the delay it introduced was measured on a system-call and a process level. pH adds a delay of $4.7\mu\text{s}$ to system-calls that would either take less than $2\mu\text{s}$. As far as processes go, the kernel build took twice as long to finish, while in the other two (`find / -print` και `Quake 2`) the overhead was not noticeable. These show that pH adds a certain amount of delay to the system, but it's up to the process how much that will be.

Discussion

In conclusion, this paper has shown us that:

- It is feasible to use system call delays to stop unknown attacks
- In order for the pH to be effective stable normal profiles must be obtained
- System call monitoring is practical with little perceptible overhead

Still, pH is not completely secure. There are no checks to ensure that a profile has not been tampered with on disk. An attacker could design a less-detectable attack based on the system call usage on the target machine and pH could be used to generate a DOS attack by triggering abnormal behavior in a target program.

Finally, some large programs are implemented using user space threads, causing system-calls to be interleaved, thus the system call profiles of these programs may never stabilize.