Automated Response Using System-Call Delays

by Anil Somayaji and Stephanie Forrest

Proceedings of the 9th USENIX Security Symposium
Intrusion Detection System (IDS)

➔ A device or a software application.

➔ Monitors network or system activities for malicious activities or policy violations.

➔ Produces reports to a management station.

source: http://en.wikipedia.org/wiki/Intrusion_detection_system
Computer Security has focused mostly on

- prevention
  cryptography, firewalls, protocol design

- detection
  virus, intrusion detection

- response
And there arises the problem of...

the automated response!
Good anomaly detection comes at the price of persistent false positives.
Reasons of false-positives

➔ Dynamic environments.

➔ Profiles of legitimate activity change over time.

➔ Ambiguity in the distinction between normal and abnormal activities.
Approaching the automated response...

Need a system that

➔ autonomously monitors its own activities

➔ routinely makes small corrections to maintain homeostasis
The System: Process Homeostasis (pH)

- a set of Linux kernel extensions
- does not interfere with normal operation
- successfully stops attacks

- connects system calls with feedback mechanisms that can **delay** and **abort** anomalous system calls
Process Homeostasis (pH)

Delays

➔ Interfering with program behavior
➔ Implemented as an increasing function of recent anomalous sequences
Process Homeostasis (pH)

Contributions

➔ With minimal performance overhead, demonstrates the feasibility of monitoring every active process at the system-call level in real-time.

➔ Introduces a practical, relatively non-intrusive method for automatically responding to anomalous program behavior.
The Idea

➔ For each process invoked, begin a new system-call trace.
➔ Collect such traces over many invocations of a program, while behaving normally.
➔ Use this collection to develop an empirical model of its normal behavior.
The Goal

➔ Categorize most normal behavior as normal.

➔ Categorize most attacks as abnormal.
pH
The Constraints

➔ On-line training and testing.
➔ Large alphabet sizes.
➔ Models sensitive to common forms of intrusion.

Traces of intrusions are often 99% the same as normal traces, with very small, temporally clumped deviations from normal behavior.
pH
The Traces

A sequence of system calls:

execve, brk, open, fstat, mmap, close, open, mmap, munmap

Small fixed size of window that slides over each trace: 4
### pH

**The Traces**

<table>
<thead>
<tr>
<th>current</th>
<th>position 1</th>
<th>position 2</th>
<th>position 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>execve</td>
<td>execve</td>
<td>brk</td>
<td>brk</td>
</tr>
<tr>
<td>brk</td>
<td>brk</td>
<td>open</td>
<td>open</td>
</tr>
<tr>
<td>open</td>
<td>fstat</td>
<td>mmap</td>
<td>mmap</td>
</tr>
<tr>
<td>fstat</td>
<td>mmap</td>
<td>close</td>
<td>close</td>
</tr>
<tr>
<td>mmap</td>
<td>close</td>
<td>open</td>
<td>mmap</td>
</tr>
<tr>
<td>close</td>
<td>open</td>
<td>mmap</td>
<td>munmap</td>
</tr>
</tbody>
</table>

Join the lines with the same current value & compress.

Store this table using a fixed-size bit array \(|S| \times |S|\).
About Mismatches

➔ System call pair:
   The current call and a preceding call within the current window.

➔ Mismatch:
   Any system call pair not present in the normal profile.

➔ True positives:
   When a mismatch indicates actual anomalous behavior.

➔ False positives:
   When a mismatch is a sequence that was not included in the normal training data.
About Mismatches

- System call pair:
The current call and a preceding call within the current window.

- Mismatch:
Any system call pair not present in the normal profile.

- True positives:
When a mismatch indicates actual anomalous behavior.

- False positives:
When a mismatch is a sequence that was not included in the normal training data.

The current system call is defined as anomalous if there are any mismatches within its window.
pH Design

The Functions

Monitors individual processes at the system-call level.

Automatically responds to anomalous behavior (delays or aborts).
pH Design
Deep in the Kernel

- Minimize I/O requirements
- Maximize efficiency
- Mechanisms in the system call dispatcher
- Secure implementation
The Arrays

Training Array

➔ Continuously updated with new pairs.
➔ A candidate future normal profile.

Testing Array

➔ Used to detect anomalies.
➔ Never modified- only replaced with a copy of the training array.
➔ The current normal profile.
The 3 conditions to the replacement (training → testing):

1. The users signals that the profile’s training is valid (sys_pH).
2. The profile anomaly count exceeds the parameter anomaly_limit.
3. The training formula is satisfied.
pH Design
The Delays

pH responds to anomalies by delaying system call execution.

➔ The delay is $d = 2^{\text{LFC}}$ (LFC for Locality Frame Count).
   ◆ If the LFC exceeds the tolerization_limit parameter, the training array is reset, preventing truly anomalous behavior into the testing array.

➔ The scaled delay is $d \times \text{delay_factor}$.
   ◆ delay_factor: user defined parameter can be set to 0
**pH Implementation**

- Program profiles stored on the disk (load and write).
- Training and testing arrays contained in each profile.
- Two system calls added (pH_process_syscall & pH_sys).
pH Implementation
Program profiles stored on the disk

➔ On execve, kernel loads the current profile.
➔ If it is not present, a new one is created.
➔ When another process runs the same executable, the profile is shared.

➔ On termination, kernel writes the profile on the disk.
➔ A loaded profile consumes ~80K of kernel memory.
pH Implementation

➔ Program profiles stored on the disk (load and write).

➔ Training and testing arrays contained in each profile.

➔ Two system calls added (pH_process_syscall & pH_sys).
pH Implementation
Training and testing arrays contained in each profile

A profile is a structure that contains:

➔ two byte-arrays for storing pairs
➔ additional training statistics
pH Implementation

➔ Program profiles stored on the disk (load and write).
➔ Training and testing arrays contained in each profile.
➔ Two system calls added (pH_process_syscall & pH_sys).
pH Implementation

Two system calls added (pH_process_syscall & pH_sys)

The pH_process_syscall implements:

1. the monitoring
2. the response
3. the training logic
pH Implementation

Two system calls added (pH_process_syscall & pH_sys)

The pH_sys allows the superuser to:

- start, stop monitoring processes
- set system parameters
- turn on/off logging of system calls to disk (debugging)
- turn on/off logging novel sequences to disk
- print the status of system parameters
- write profiles to disk
- reset a profile
- copy the training array to its testing array (profile marked normal)
- tolerize
- clear the training array
- turn on/off debugging messages sent to kernel logging facility
pH Experimental Results
The 3 Aspects of the System

➔ Can pH detect and stop attacks in time to prevent system compromise?

➔ What is the overhead of running pH?

➔ How is pH in practice?
Three violations:

1. SSH daemon backdoor
2. SSH daemon buffer overflow
3. Sendmail attack that exploits a bug in the kernel
pH Experimental Results

The 3 Aspects of the System

Three violations:
1. SSH daemon backdoor
2. SSH daemon buffer overflow
3. sendmail attack that exploits a bug in the kernel
pH Experimental Results
The 3 Aspects of the System
delays & execve aborts
pH Experimental Results

The 3 Aspects of the System

delays & execve aborts

STOPS ATTACKS
pH Experimental Results

The 3 Aspects of the System

➔ Can pH detect and stop attacks in time to prevent system compromise?

➔ What is the overhead of running pH?

➔ How is pH in practice?
pH Experimental Results

The 3 Aspects of the System

<table>
<thead>
<tr>
<th>System Call</th>
<th>Standard ((\mu s))</th>
<th>pH ((\mu s))</th>
</tr>
</thead>
<tbody>
<tr>
<td>getpid</td>
<td>1.1577 (0.000000)</td>
<td>5.8898 (0.00025)</td>
</tr>
<tr>
<td>getusage</td>
<td>1.9145 (0.000000)</td>
<td>6.6137 (0.00138)</td>
</tr>
<tr>
<td>gettimeofday</td>
<td>1.6703 (0.00184)</td>
<td>6.3779 (0.00112)</td>
</tr>
<tr>
<td>sigaction</td>
<td>2.5609 (0.00010)</td>
<td>7.2928 (0.01029)</td>
</tr>
<tr>
<td>write</td>
<td>1.4135 (0.00187)</td>
<td>6.1637 (0.00075)</td>
</tr>
</tbody>
</table>

System call latency results.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Standard ((\mu s))</th>
<th>pH ((\mu s))</th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td>408.80 (00.618)</td>
<td>2497.90 (40.923)</td>
</tr>
<tr>
<td>simple</td>
<td>2396.24 (11.124)</td>
<td>8206.62 (11.795)</td>
</tr>
<tr>
<td>/bin/sh</td>
<td>9385.66 (26.761)</td>
<td>18223.96 (26.777)</td>
</tr>
</tbody>
</table>

Dynamic process creation latency results.

**Standard:** Linux 2.2.14 kernel.

**pH:** Linux 2.2.14 kernel with pH extensions, monitoring enabled for all processes & status messages and automated response off.

Although these tables show a significant performance hit, they are not indicative of the impact on overall system performance.

If delays are turned off, a user can use the modified workstation without noticing any differences in system behavior.
pH Experimental Results

The 3 Aspects of the System

➔ Can pH detect and stop attacks in time to prevent system compromise?

➔ What is the overhead of running pH?

➔ How is pH in practice?
Privileged programs (e.g. login, sendmail) tend to settle into a stable normal and exhibit a few anomalies.

Non-privileged programs (e.g. emacs) tend not to shift into a normal monitoring mode, are never delayed.
pH Experimental Results
The 3 Aspects of the System

→ Simple system monitoring programs (e.g. wmapm) execute a large number of system calls, have repetitious behavior, obtain normals.
→ A large privileged program, X server, tends to acquire a normal profile quickly.
Simple system monitoring programs (e.g. wmapm) execute a large number of system calls, have repetitious behavior, obtain normals.

A large privileged program, X server, tends to acquire a normal profile quickly.

Programs which make large numbers of system calls in a short period of time tend to acquire normal profiles, even when a true sampling of behavior has not yet occurred.
The three exploits help show that pH can use system-call delays to stop intrusions in real-time, even for very different types of attacks.

There is an ongoing risk that pH could be trained to accept intrusions as normal behavior.

It may be necessary to implement a default timeout mechanism through pH, in which any process that is delayed beyond a certain point is automatically terminated.

pH in its current form, however, monitors and responds to anomalies in all programs.
Automated Response Using System-Call Delays
by Anil Somayaji and Stephanie Forrest

Eva Papadogiannaki
for
(CS-558) Internet Systems and Technologies
Figure 1: Basic flow of control and data in a pH-modified Linux kernel.