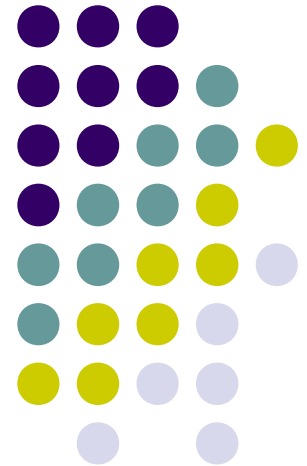


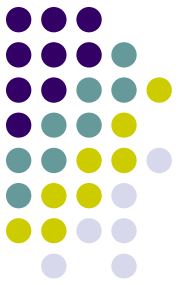
# Automated Response Using System-Call Delays

Anil Somayaji and Stephanie Forrest  
Dept. of Computer Science  
University of New Mexico  
9<sup>th</sup> USENIX Security Symposium 2000

ligouras@csd.uoc.gr  
CS455

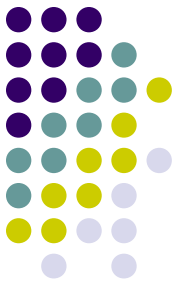


# Introduction



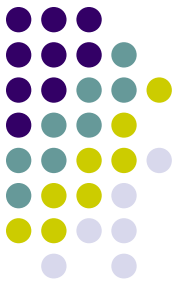
- Motivation
  - Computer security research has focused mainly on prevention and detection
  - IDSs need constant administration in order to reduce the false positives (e.g removing a legitimate user from the network)
  - Thus an automated response system can reduce administrator's workload
- Contributions
  - Monitoring every process at system-call level in real-time
  - Introducing a practical method for automatically responding to abnormal program behavior

# Introduction



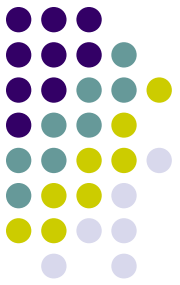
- Anomaly detection – Persistent false positives
  - Perpetual Novelty
  - Profiles of legitimate traffic change continually
  - Inherit ambiguity between normal and abnormal activities
- pH (process homeostasis)
  - Automated response system that allows a computer to maintain its own integrity
  - Implemented as a set of extensions to a Linux kernel
  - Enhances system calls with feedback mechanisms that either delay or abort anomalous system calls

# Methodology



- Monitor all system calls made by an executing program on per-process basis
- For every new process we create a new trace of system calls (in time-series)
- An empirical model of the program's normal behavior is developed using the collection of all traces

# Methodology



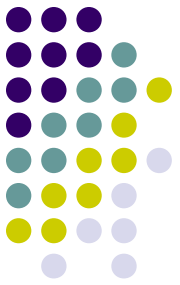
- Model Requirements
  - Suitable for online training and testing
  - Suitable for large size alphabets (number of sys calls)
  - Sensitive to common forms of intrusion (<1% of the trace is abnormal)

# Methodology



- Time-delay embedding
  - The normal behavior is defined in terms of short n-grams of system calls
  - A sliding window is used over every trace in order to find all the “pairs” (current system call , previous system call).
  - The collection of unique pairs over all traces for a single program constitutes a model of normal behavior for the program

# Examples

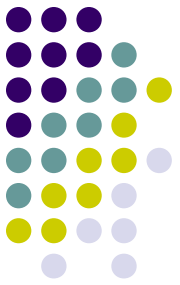


execve, brk, open, fstat, mmap, close, open,  
mmap, munmap

position 3	position 2	position 1	current
			execve
		execve	brk
	execve	brk	open
execve	brk	open	fstat
brk	open	fstat	mmap
open	fstat	mmap	close
fstat	mmap	close	open
mmap	close	open	mmap
close	open	mmap	munmap

Window size=4

# Examples



This table can be stored in a  $|S| \times |S| \times (w-1)$  bit-array

position 3	position 2	position 1	current
fstat	execve, mmap	execve	execve
execve	brk	brk, close	brk
brk, mmap	open, close	open	open
open	fstat	fstat, open	fstat
close	open	mmap	mmap
		mmap	close
			munmap

# Definitions



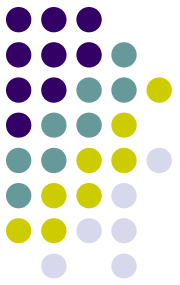
- Mismatch

- Any system call pair not present in the normal profile is called a mismatch.
- Any mismatch could be a true or a false positive
- The current system call is defined as anomalous if there are any mismatches within its window.

- Locality Frame

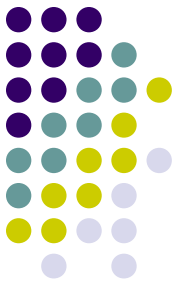
- A circular array is used to hold the recent anomalous calls
- Let  $A_i$  be the  $i$ -th entry of the locality frame with  $0 \leq i < n$  and  $A_i \in \{0,1\}$ . For system call  $s$  with mismatches  $m_s$ ,  $A_{s \bmod n} = 1$  if  $m_s > 0$  and 0 otherwise. The total of recent anomalies,  $\sum A_i$ , is called the locality frame count LFC

# pH Design



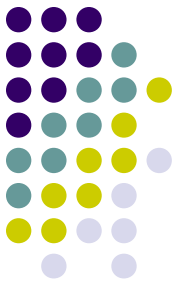
- pH performs two important operations:
  - pH monitors individual processes at the system-call level
  - Automatically responds to anomalous behavior by either slowing down or aborting system calls
- pH in Kernel space
  - Minimize I/O requirements and maximize efficiency, stability and security
  - Alternative approaches have serious drawbacks
    - Audit packages
    - User-space tracing utilities
    - Instrumented libraries

# pH Design



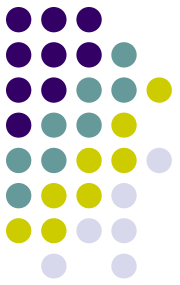
- pH maintains two arrays of data:
  - Training array: Continuously update with the system calls as they appear
  - Testing array: Current 'normal' model may be replaced by training array
- The replacement of the testing array occurs under three circumstances
  - via a special system call (`sys_pH`)
  - The profile anomaly count exceeds the parameter `anomaly_limit`;
  - The training formula is satisfied.

# pH Design



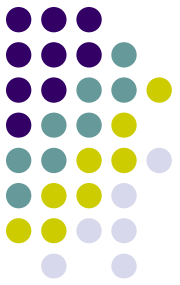
- The training array is copied onto the testing array if it meets all the following conditions :
  - $\text{last\_mod\_count} > \text{mod\_minimum}$
  - $\text{normal\_count} > \text{normal\_minimum}$
  - $\text{train\_count}/\text{normal\_count} > \text{normal\_ratio}$
- Where:
  - $\text{train\_count} = \# \text{calls since array initialization}$
  - $\text{last\_mod\_count} = \# \text{calls since array was last modified}$
  - $\text{normal\_count} = \text{train\_count} - \text{last\_mod\_count}$

# pH Design



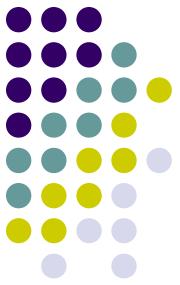
- pH responds to anomalies by delaying system call execution
  - Delay for a system call =  $d \times \text{delay\_factor}$  ( $d=2^{\text{LFC}}$ )
- If  $\text{LFC} > \text{tolerization\_limit}(=12)$  the training array is reset
- If  $\text{maxLFC} > \text{abort\_execve}$  all `execve`'s are abort for the current process
- Tolerization

# Implementation

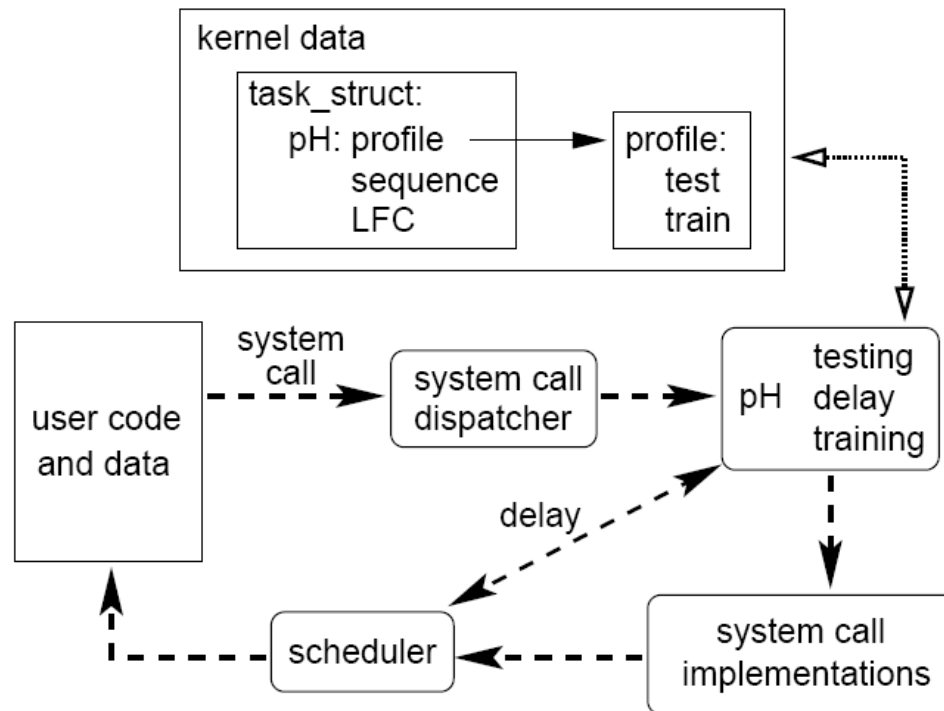


- The pH is implemented as a patch for the Linux 2.2 kernel
- Program profiles are stored on disk. Each profile contains both a training and a testing array
- The kernel loads the program's profile when it begins execution
- A loaded profile consumes  $< 80K$

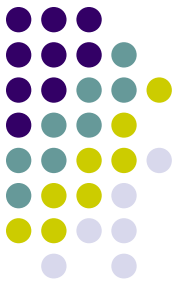
# Implementation



The system call dispatcher is modified so that it calls `pH_process_syscall` which implements the monitoring, response and training logic.

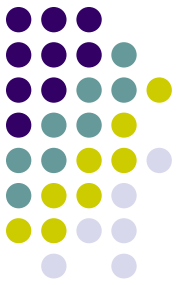


# Experimental Results

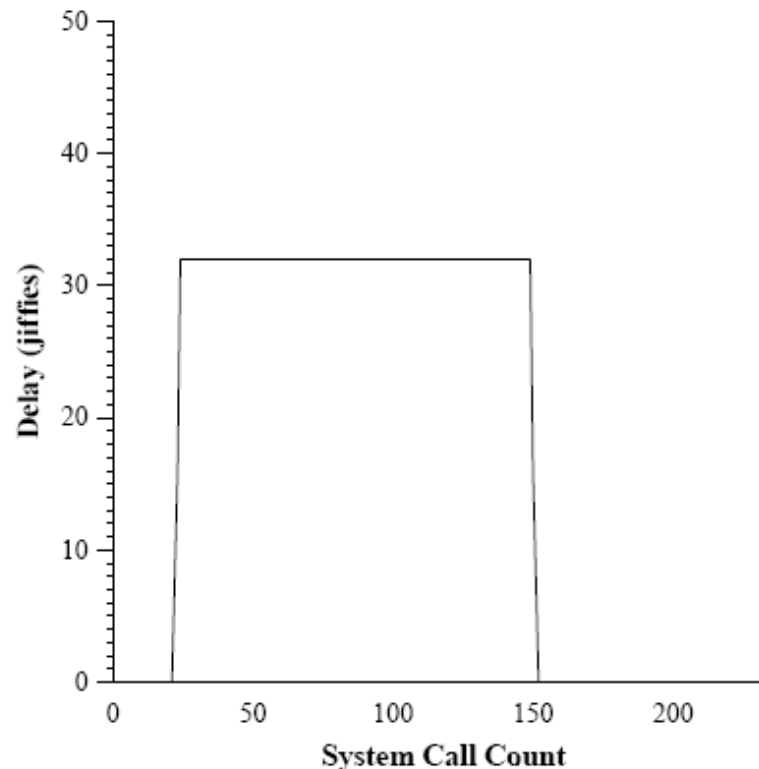


- Three security violations were used
  - An SSH daemon backdoor
    - A normal profile of 1725 pairs was created
    - It produced 5 anomalies (3 to the parent and 2 to the child)
  - An SSH daemon buffer-overflow
  - A sendmail attack that exploits a bug into linux kernel's capabilities code

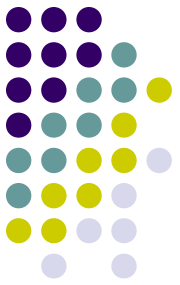
# Experimental Results



- Setting *delay\_factor* to 4 the resulting connection was slowed down significantly
- With *abort\_execve* set to 1, the backdoor was closed

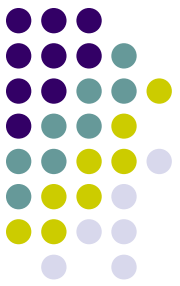


# Experimental Results



- The buffer overflow attack
  - Produced 4 anomalies.
  - Setting `delay_factor` to 4 produced the same anomalies but produced at least an 80s delay
  - With `execve` aborts enabled, the overflow attack was stopped
- The sendmail exploit
  - A normal profile was created with 2412 system calls pairs
  - Numerous anomalies caused the `tolerization_limit` to be reached numerous times.
  - Enabling `execve` aborts did nothing to inhibit the attack.
  - A `delay_factor` of 4 effectively stopped the attack since delays of at least two hours were produced

# Experimental Results



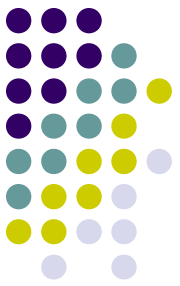
System Call	Standard ( $\mu s$ )	pH ( $\mu s$ )
getpid	1.1577 (0.00000)	5.8898 (0.00025)
getrusage	1.9145 (0.00000)	6.6137 (0.00138)
gettimeofday	1.6703 (0.00184)	6.3779 (0.00112)
sigaction	2.5609 (0.00010)	7.2928 (0.01029)
write	1.4135 (0.00187)	6.1637 (0.00075)

Table 1: System call latency results. All times are in microseconds. Standard deviations are listed in parentheses.

Operation	Standard ( $\mu s$ )	pH ( $\mu s$ )
null	408.80 (00.618)	2497.90 (40.923)
simple	2396.24 (11.124)	8206.62 (11.795)
/bin/sh	9385.66 (26.761)	18223.96 (26.777)

Table 2: Dynamic process creation latency results. Null refers to a fork of the current process. Simple is a fork of the current process plus an `exec()` of a hello-world program written in C. `/bin/sh` refers to the execution of hello-world through the `libc system()` interface, which uses `/bin/sh` to invoke hello-world. All times are in microseconds. Standard deviations are listed in parentheses.

# Experimental Results



Benchmark	Standard	pH
kernel build (s)		
real	702.47 (0.07)	727.44 (0.29)
user	669.35 (0.60)	673.67 (0.55)
sys	33.00 (0.61)	53.60 (0.70)
find / -print (s)		
real	5.68 (0.58)	6.24 (0.54)
user	1.61 (0.09)	1.59 (0.09)
sys	3.27 (0.09)	3.90 (0.17)
Quake 2 (fps)		
demo1	22.89 (0.03)	22.87 (0.05)
demo2	23.30 (0.00)	23.30 (0.00)

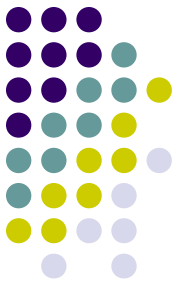
Table 3: Overall system performance. All units are seconds (s), except for the Quake 2 test, which is in frames per second (fps). Ten trials were run for each example, except 100 trials were run for `find`. Each test was run once before beginning the measurements in order to eliminate initial I/O transients. Standard deviations are listed in parentheses.

# Discussion



- 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

# Discussion



- 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
  - pH could be used to generate a DOS attack by triggering abnormal behavior in a target program
- 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

# The End

- Questions?

