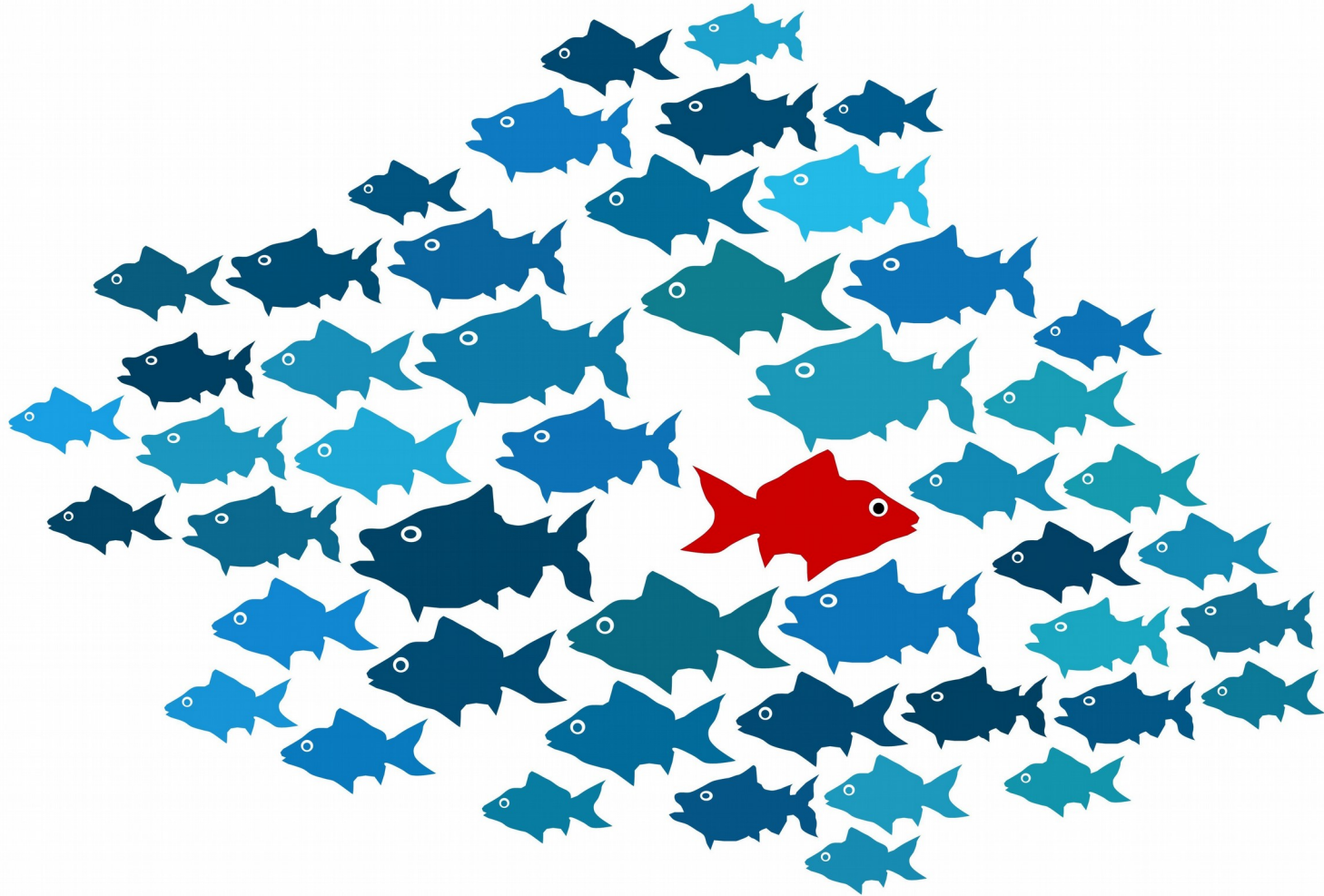


A Novel approach to anomaly detection using system-call sequences

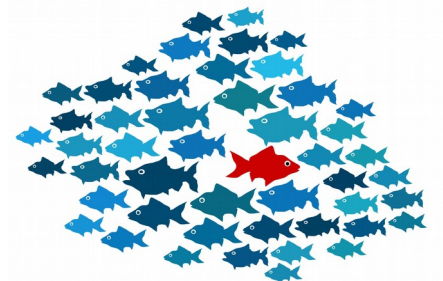


Motivation

- Security walls have proven to be an insufficient security measure (crypto, firewall, access control, etc.)
- Systems are a complex aggregate of interconnected systems. Integrators get black box components from manufacturers.
- The son-of-Eve is usually the weakest link in the security chain e.g:
 - Incorrect or non-existent security policies
 - Unsafe/insecure software req. gathering/design/implementation/test
 - Formal verification and sound software engineering are not enough in this ever-changing ecosystem.

Anomaly detection using short sequences of system calls

- Look for an anomalous pattern of behavior and sound react accordingly (**without a clear model of an attack pattern**)
- The idea is to catch **zero**-day attacks (stuff we haven't seen before)



Main components

- Defining NORMAL by building a profile (training)
- Monitoring behavior and detecting anomaly
- Reacting (Sounding the alarm in this case)

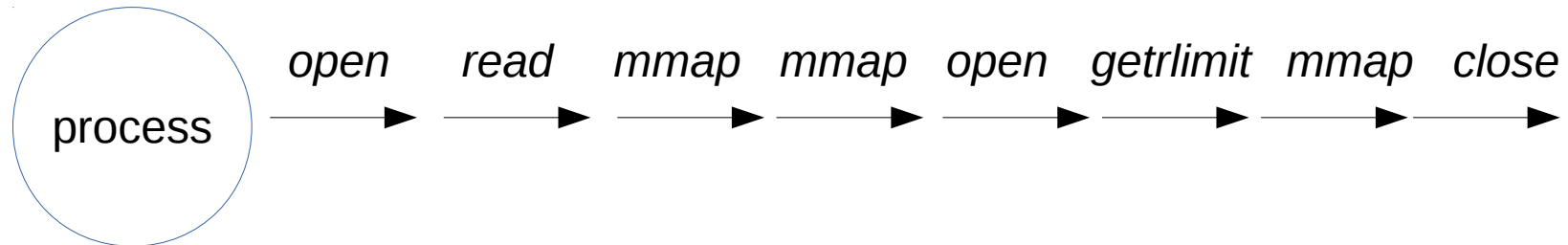
Determine the correct behavior of a process by building a profile of the unique sequences of system calls it emits.

- Treating the process as a black-box that emits signals and can be watched
- Evidence (in academic literature) suggest that **short-sequences of system calls** are a good discriminator between abnormal and normal behavior of a process. They generate a stable **SIGNATURE** of normal behavior.


(The signature has low variance over a wide range of normal operating conditions and has a high probability of being perturbed when abnormal activities, such as attacks or attack attempts, occur)

- **Only the temporal ordering of the system calls** are of concern for now (looking into other parameters can be added later if needed)
- Detection is distributed (separate and independent for each process)
- Logically: a very narrow profile will result in too many false positives and vice versa a too broad of a definition will result in false negatives (allowance of bad behavior)
- Very simple and suitable for an **ON-LINE RUNTIME IDS**


- For a non-trivial program, theoretical sets of system calls will be huge. The local (short range) ordering of system calls is remarkably consistent.
- Slide a window (size $k + 1$) across the trace of system calls and record which calls follow which **within** the sliding window.



1. Contd.


window size 3 

open, *read, mmap, mmap*, *open, getrlimit, mmap, close*


Time


call	Pos 1	Pos 2	Pos 3
<i>open</i>			

1. Contd.


window size 3 

open, *read, mmap, mmap*, *open, getrlimit, mmap, close*


Time


call	Pos 1	Pos 2	Pos 3
<i>open</i>			
	<i>read</i>		

1. Contd.


window size 3 

open, *read, mmap, mmap*, *open, getrlimit, mmap, close*


Time


call	Pos 1	Pos 2	Pos 3
<i>open</i>			
	<i>read</i>		
		<i>mmap</i>	

1. Contd.


window size 3 

open, *read, mmap, mmap*, *open, getrlimit, mmap, close*

Time


call	Pos 1	Pos 2	Pos 3
<i>open</i>			
	<i>read</i>		
		<i>mmap</i>	
			<i>mmap</i>


1. Contd.

window size 3 

*open, **read**, mmap, mmap, open, getrlimit, mmap, close*

call	Pos 1	Pos 2	Pos 3
<i>open</i>	<i>read</i>	<i>mmap</i>	<i>mmap</i>

1. Contd.

window size 3 

*open, **read**, mmap, mmap, open, getrlimit, mmap, close*

call	Pos 1	Pos 2	Pos 3
<i>open</i>	<i>read</i>	<i>mmap</i>	<i>mmap</i>
<i>read</i>	<i>mmap</i>	<i>mmap</i>	

1. Contd.

window size 3 

open, read, mmap, mmap, open, getrlimit, mmap, close

call	Pos 1	Pos 2	Pos 3
<i>open</i>	<i>read</i>	<i>mmap</i>	<i>mmap</i>
<i>read</i>	<i>mmap</i>	<i>mmap</i>	
<i>mmap</i>	<i>mmap</i>		

1. Contd.

window size 3 

*open, read, mmap, mmap, open, **getrlimit, mmap, close***

call	Pos 1	Pos 2	Pos 3
<i>open</i>	<i>read getrlimit</i>	<i>mmap</i>	<i>mmap close</i>
<i>read</i>	<i>mmap</i>	<i>mmap</i>	<i>open</i>
<i>mmap</i>	<i>mmap</i>	<i>open</i>	<i>getrlimit</i>
	<i>open</i>	<i>getrlimit</i>	<i>mmap</i>
	<i>close</i>		
<i>getrlimit</i>	<i>mmap</i>	<i>close</i>	
<i>close</i>			

Sequences compressed

2. Use the profile(s) to monitor process for significant deviations from normal

- The same method is used. We slide a window across the incoming trace and determine if the incoming sequence differs from the recorded one.

open, read, mmap, open, open, getrlimit, mmap, close

call	Pos 1	Pos 2	Pos 3
<i>open</i>	<i>read</i> <i>getrlimit</i>	<i>mmap</i>	<i>mmap</i> <i>close</i>
<i>read</i>	<i>mmap</i>	<i>mmap</i>	<i>open</i>
<i>mmap</i>	<i>mmap</i>	<i>open</i>	<i>getrlimit</i>
	<i>open</i>	<i>getrlimit</i>	<i>mmap</i>
	<i>close</i>		
<i>getrlimit</i>	<i>mmap</i>	<i>close</i>	
<i>close</i>			

mmap changing to open generates

4 mismatches!

- open is not followed by open at pos 3
- read is not followed by open at pos 2
- open is not followed by open at pos 1
- open is not followed by getrlimit at pos 2

**4/18 mismatched =
22% mis-rate**

Can be efficiently implemented to run in $O(N)$ time (N = length of the trace)

The maximum number of pairwise mismatches for a sequence of length L with a lookahead of K is :

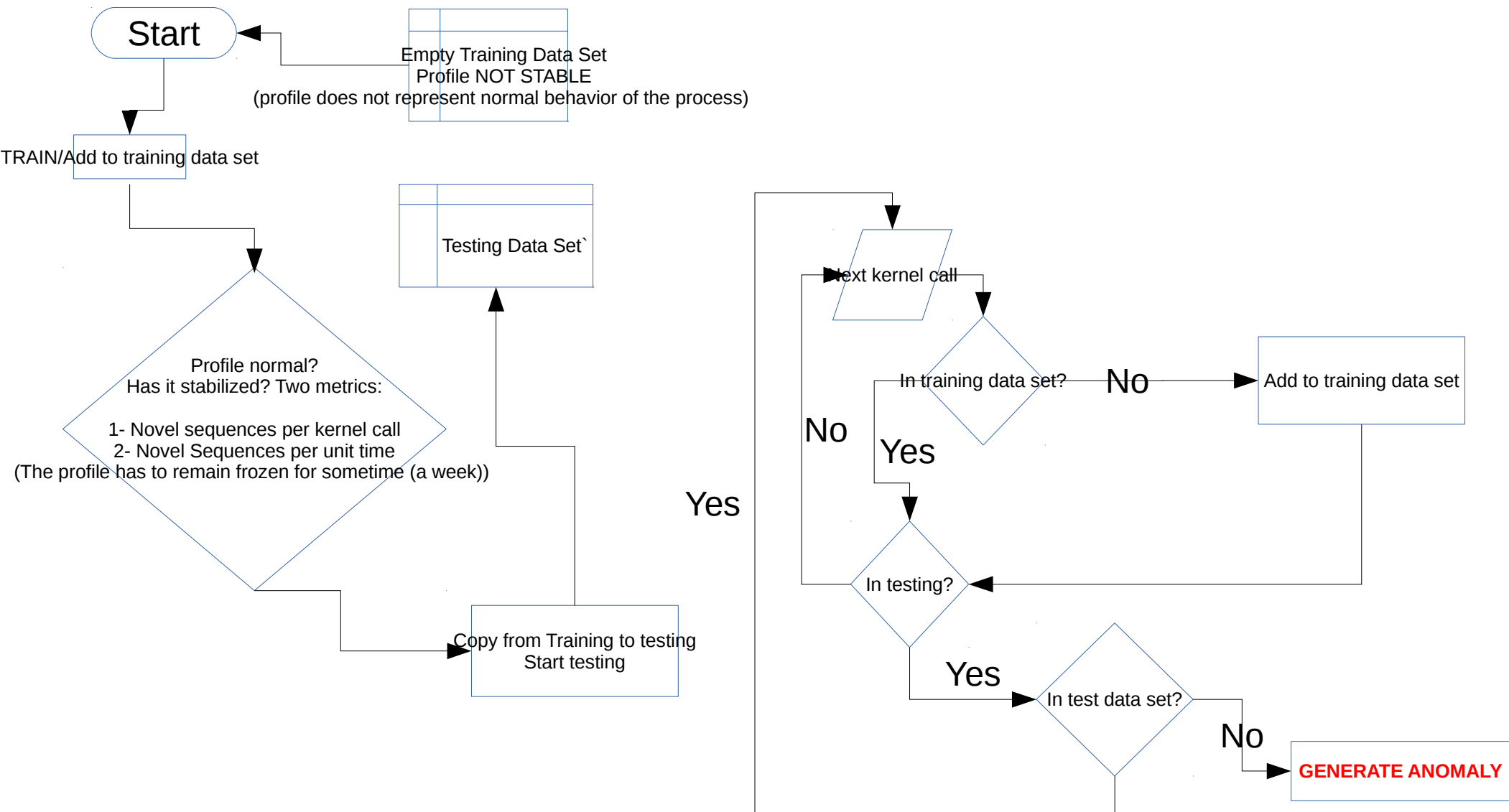
$$k(L - (k + 1) / 2)$$

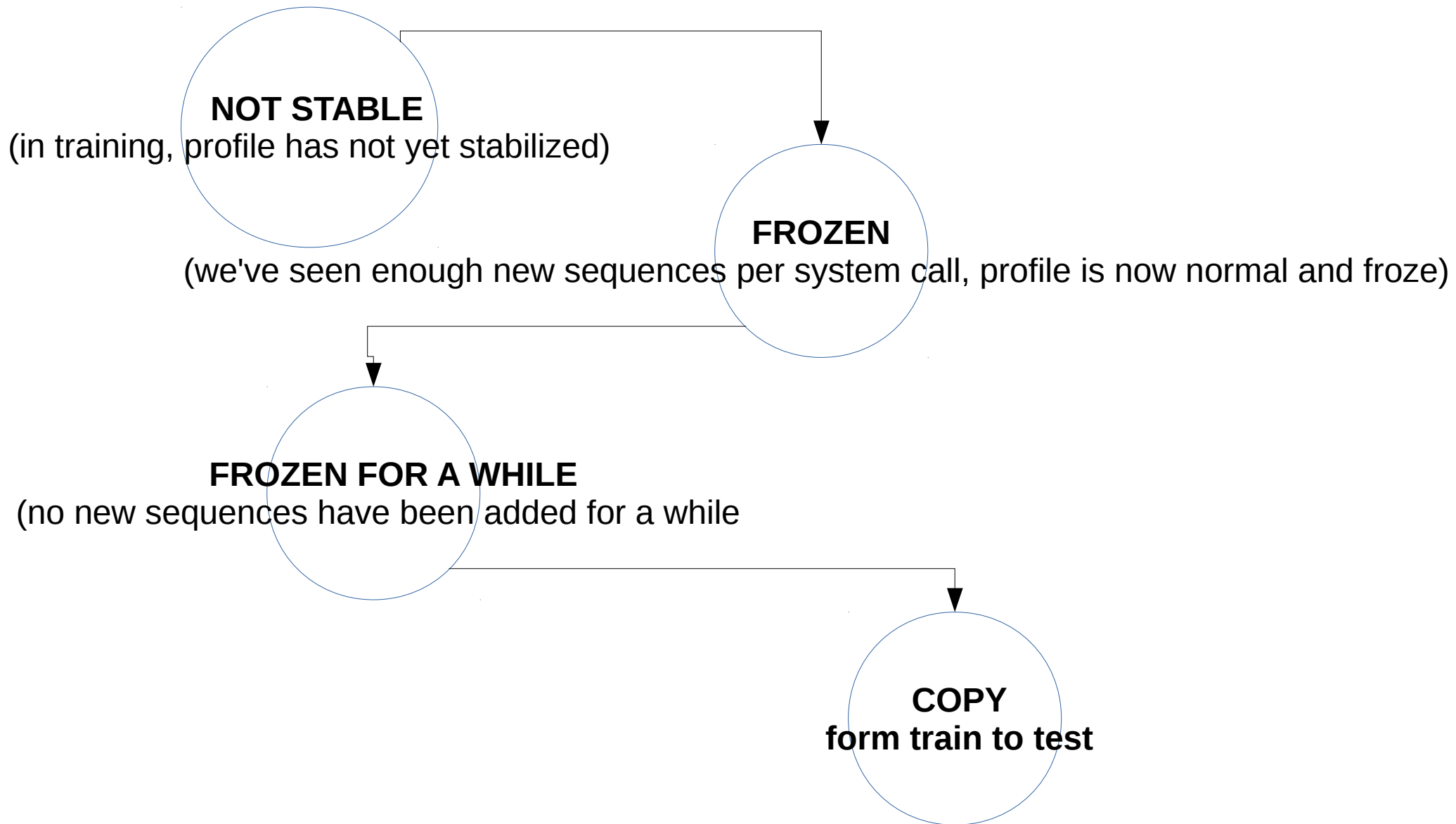
- Normal database:

- Constructed during verification phase: software is exhaustively tested and all normal variations exercised.
 - In addition, detector might be set with a time limit in which after, it stops adding mismatches to the normal profile.
 - OR detector might be set to freeze the normal profile after it has seen enough mismatches
- The size of the normal database is relatively small, making it feasible to detect mismatches at runtime. Using sendmail on a SunOS 4.1.1 (back in the day) 1.5 million system calls generated approx. 1500 entries. Not a concrete example but shows relative values.
- Threshold of mismatches must be set. A value of 0 means not tolerance of any mismatches what so ever. (I argue that this shouldn't be as big of a problem on a safety-critical system vs a server running Microsoft's IIS)

Normal behavior is a subset of all possible legal behavior

- There is a clear distinction between **normal** and **legal behavior**. Ideally, we want the normal database to contain all variations in normal behavior, but we do not want it to contain every single possible path of legal behavior.
- Unusual behavior could still be legal but not normal (e.g we ran out of disk space). If all possible execution paths are considered normal, we wouldn't be able to detect unusual conditions that are indicative of system problems.
- The larger the window size, the less generalization, the better it is for anomaly detection. The trade-off? Higher false-positives. Tweaking necessary.



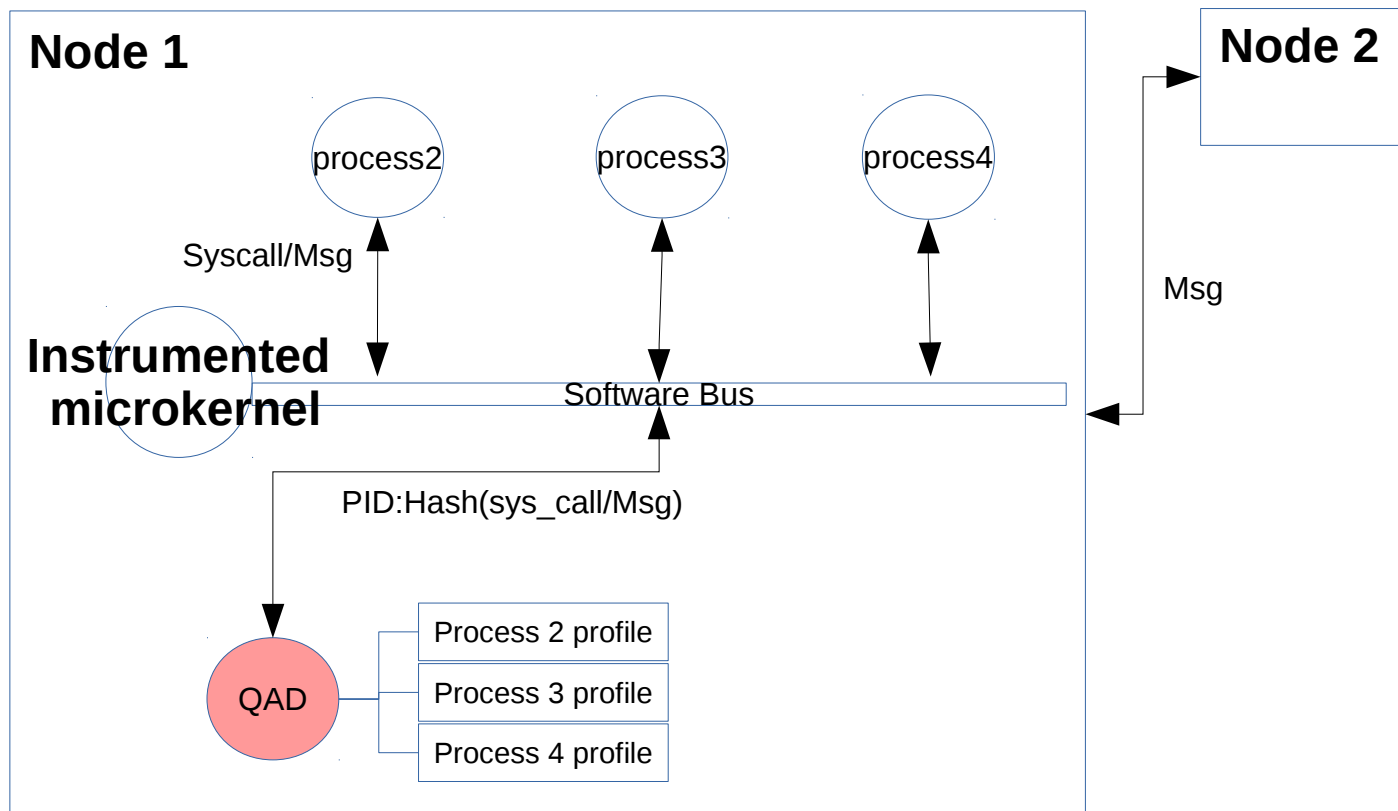


If profile is frozen and new sequences are encountered, the profile is thawed (frozen set to 0) and the new sequence is added to the training data set.

QNX System Calls

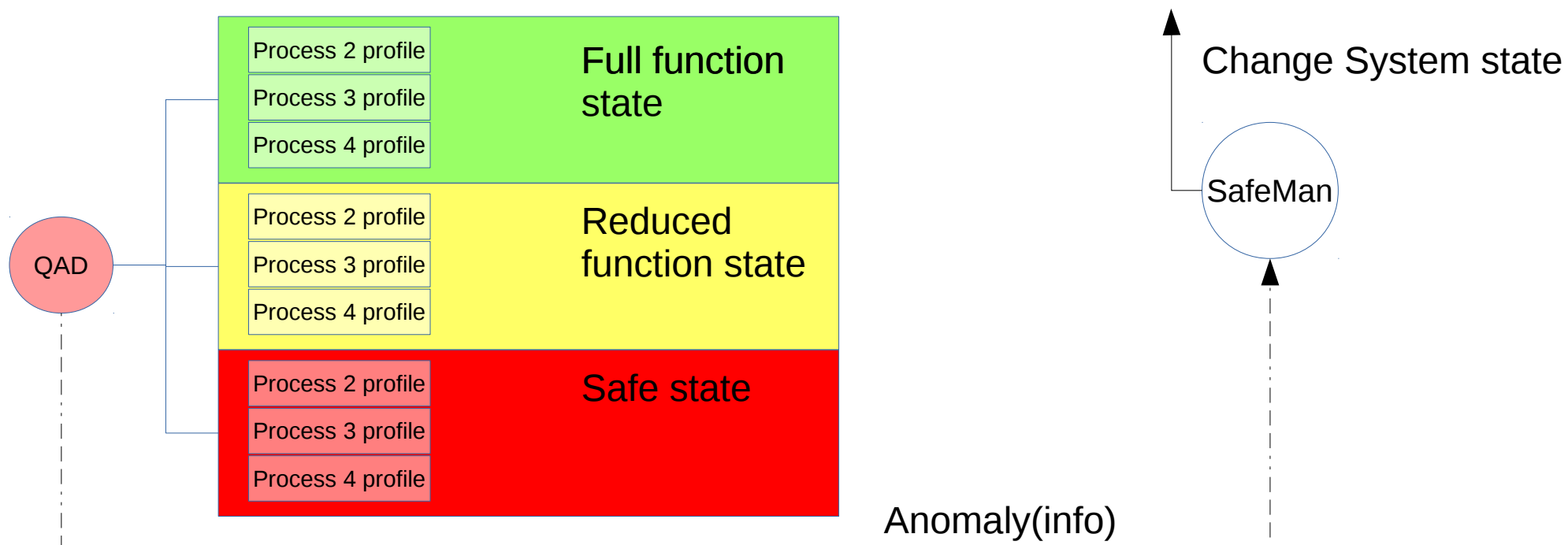
- Most sys calls are message sends
 - Can be uniquely identified by
 - PID sender
 - PID receiver
 - CHID
 - NID
 - Message head
- Hashed to produce a unique value
- A normal profile is build based on the hash of
PID snd /PID rx/CHID/NID/Message head

- This allows for detecting anomaly not just for “system calls” going to pid 0 but also for interprocess messages.
- Including those messages sent to a remote node (Node ID)



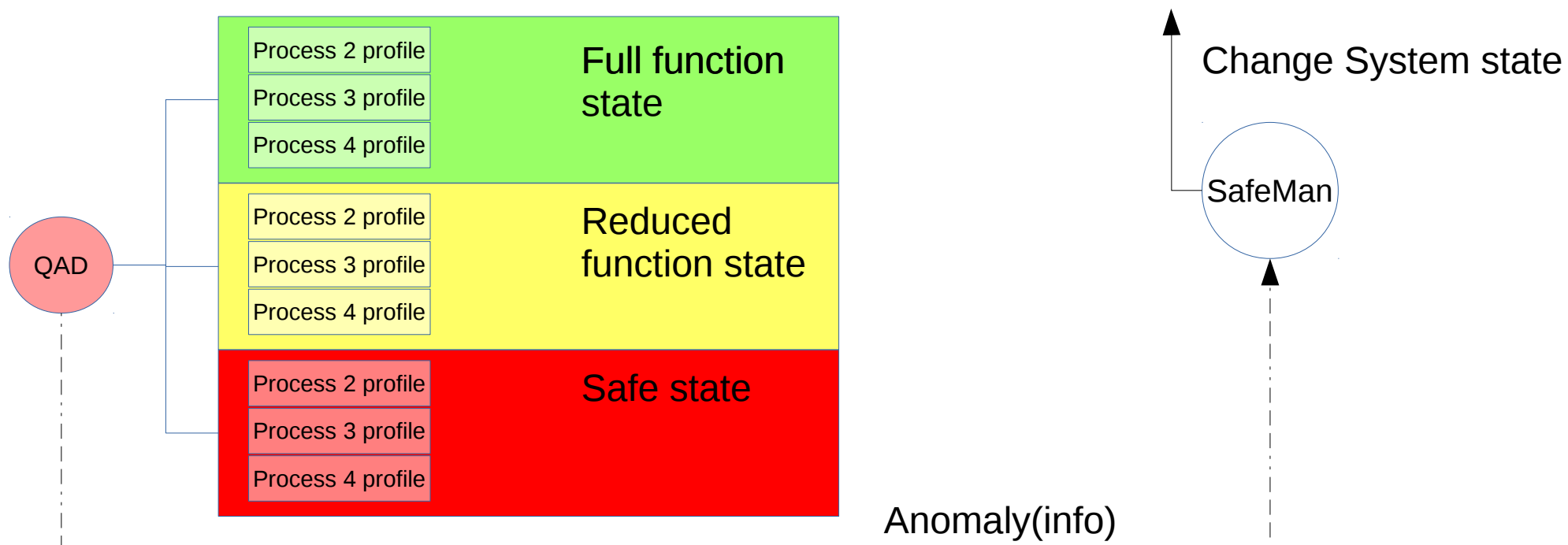
What happens when an alarm is triggered?

- Propose to get safety specialists to decide.
Perhaps a transition between safety levels? (full functional – Reduced functional – Safe state)
- Build a framework to allow customers to decide how to react?
- Learn different profiles for different safety level?



What happens when an alarm is triggered?

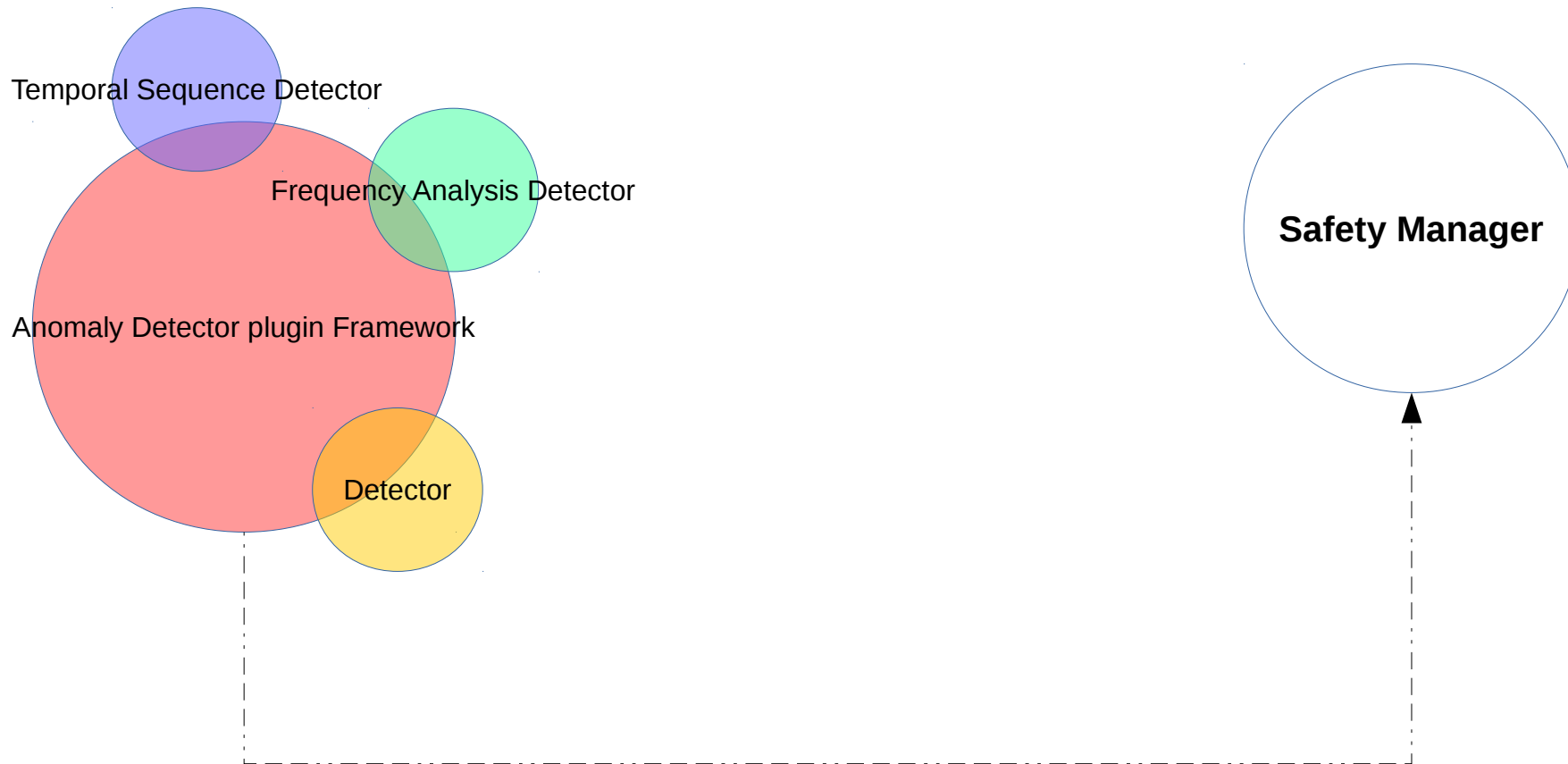
- Aborting the system call?
- Delaying the system call?
- A different framework to take care of this all.



Temporally clustered sequences

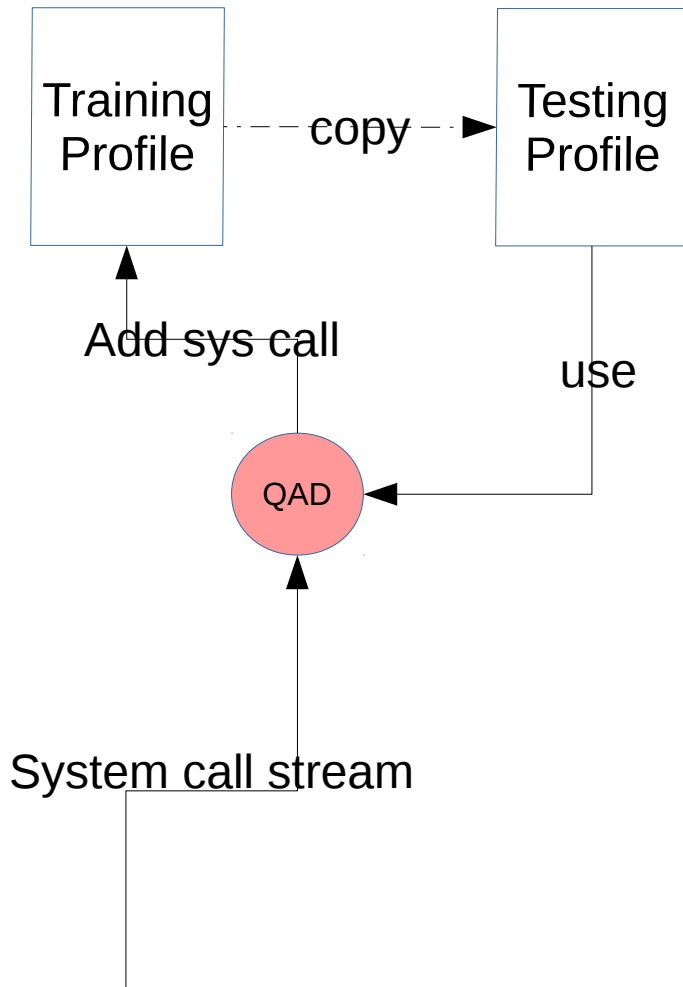
- Research has proven that anomalous sequences are produced in temporally local clusters.
- A record of the number of past n anomalous system calls is kept (locality frame count or LFC) to facilitate the detection of these clusters

Anomaly Detectors Plugin Framework



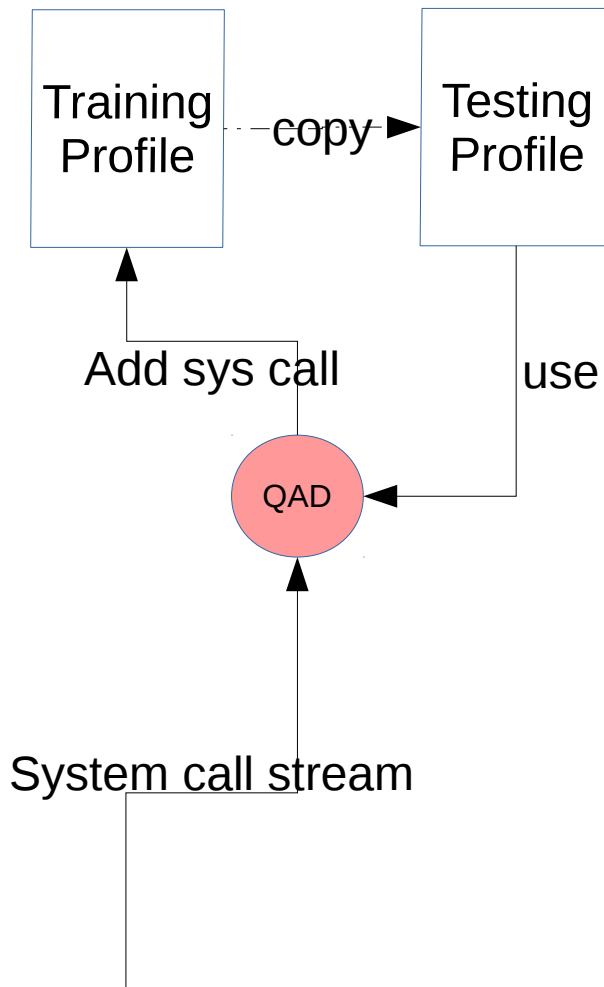
Two sets of profiles per process

- Testing array is the “current” normal.

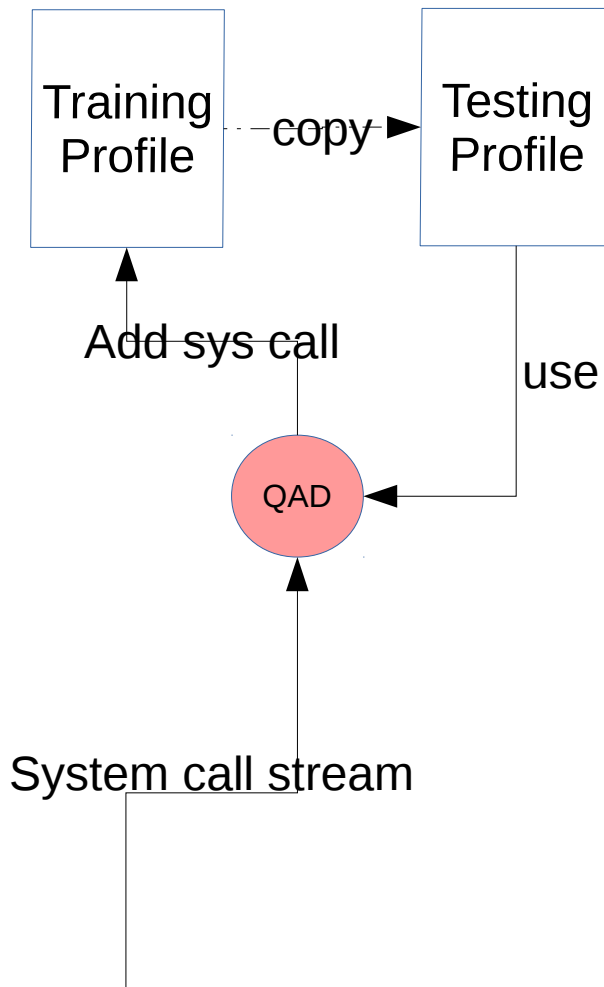


Two sets of profiles per process

- Testing profile is the “current” normal.
- Training profile is updated with new sequences (during training).



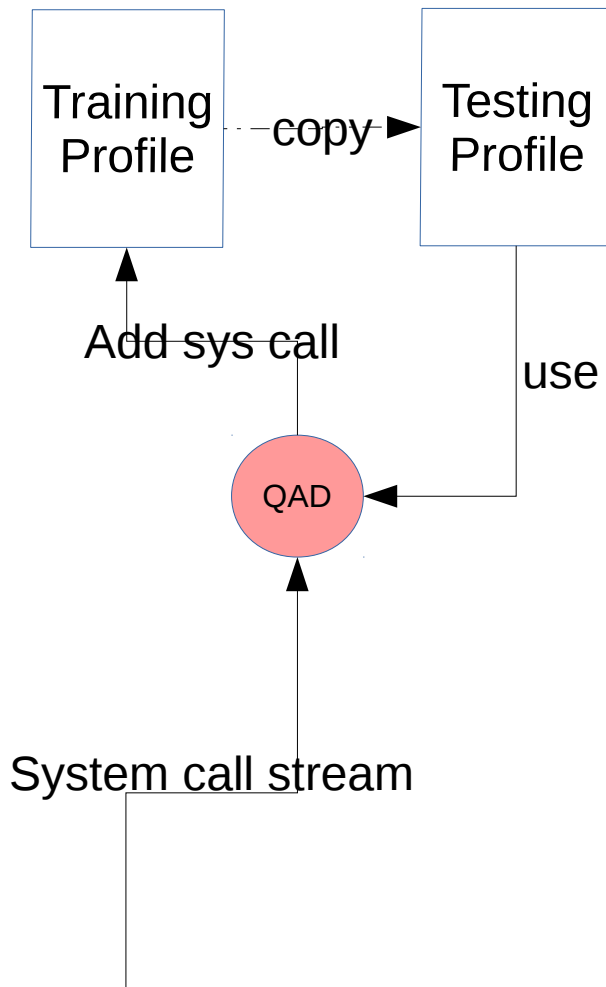
Two sets of profiles per process



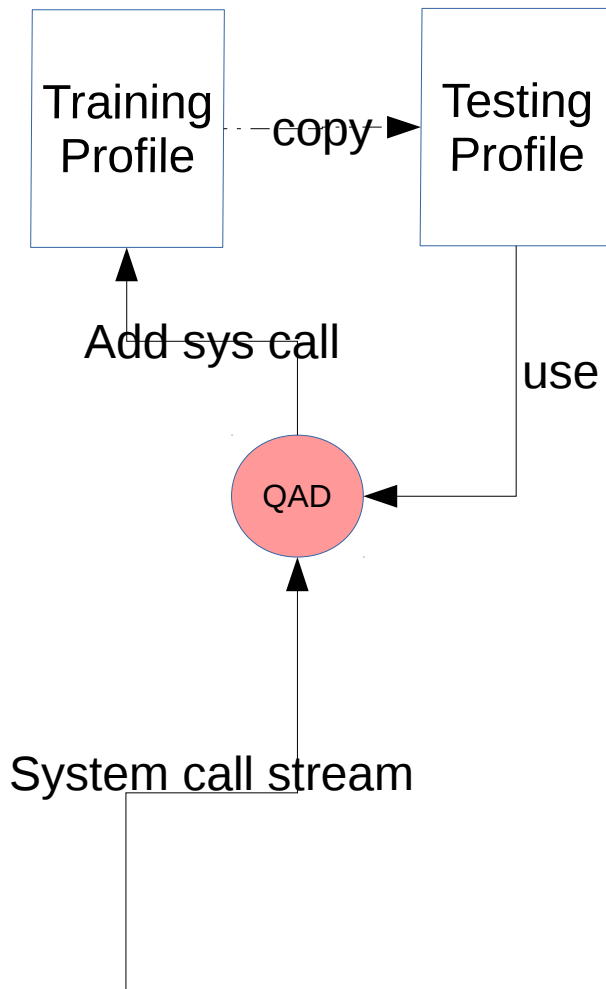
- Testing profile is the “current” normal.
- Training profile is updated with new sequences (during training).
- Testing profile is replaced by the training profile when:
 - An explicit command to the system.
 - Automatically after having seen “enough” normal behavior (no new lookahead pairs have been generated)

Two sets of profiles per process

- Testing profile is the “current” normal.
- Training profile is updated with new sequences (during training).
- Testing profile is replaced by the training profile when:
 - An explicit command to the system.
 - Automatically after having seen “enough” normal behavior (no new lookahead pairs have been generated)
 - Sometime has passed



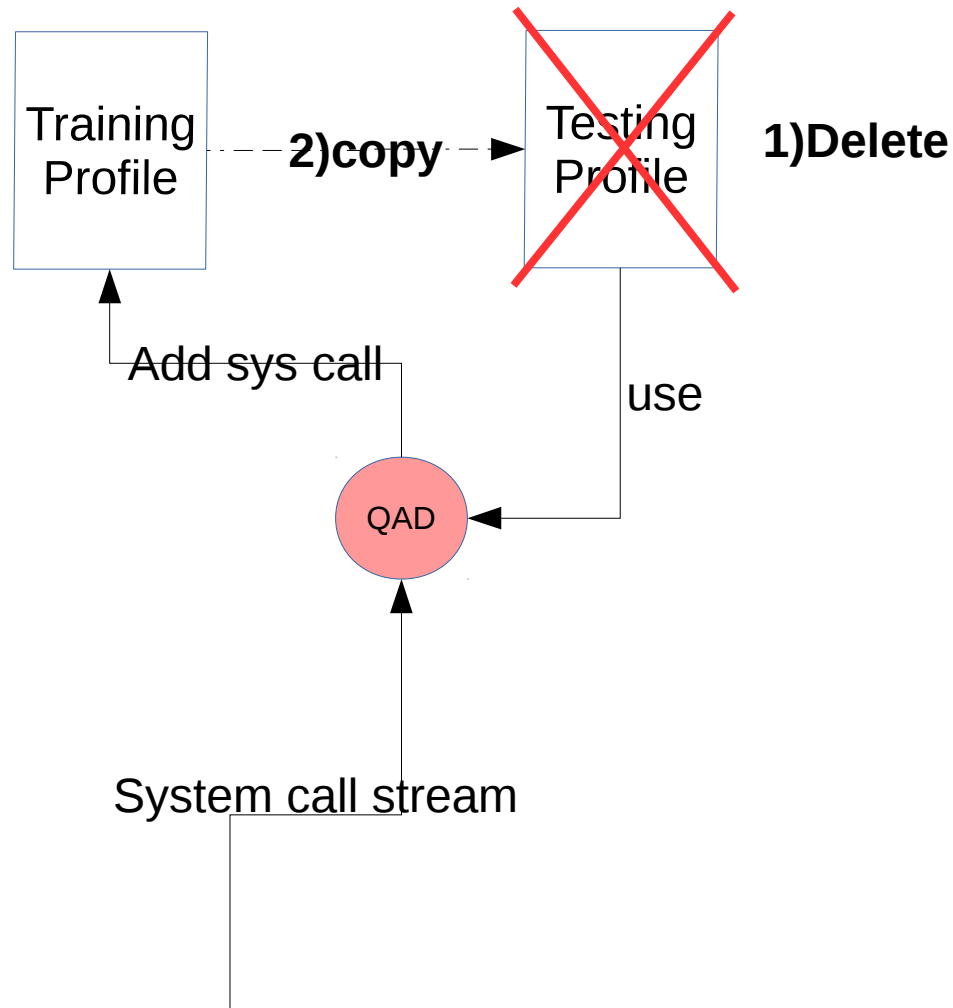
Two sets of profiles per process



- An anomaly limit value could be used after training so that new normal behavior is automatically (tolerized) or added to the training set. After which it's value is increased or blocked.

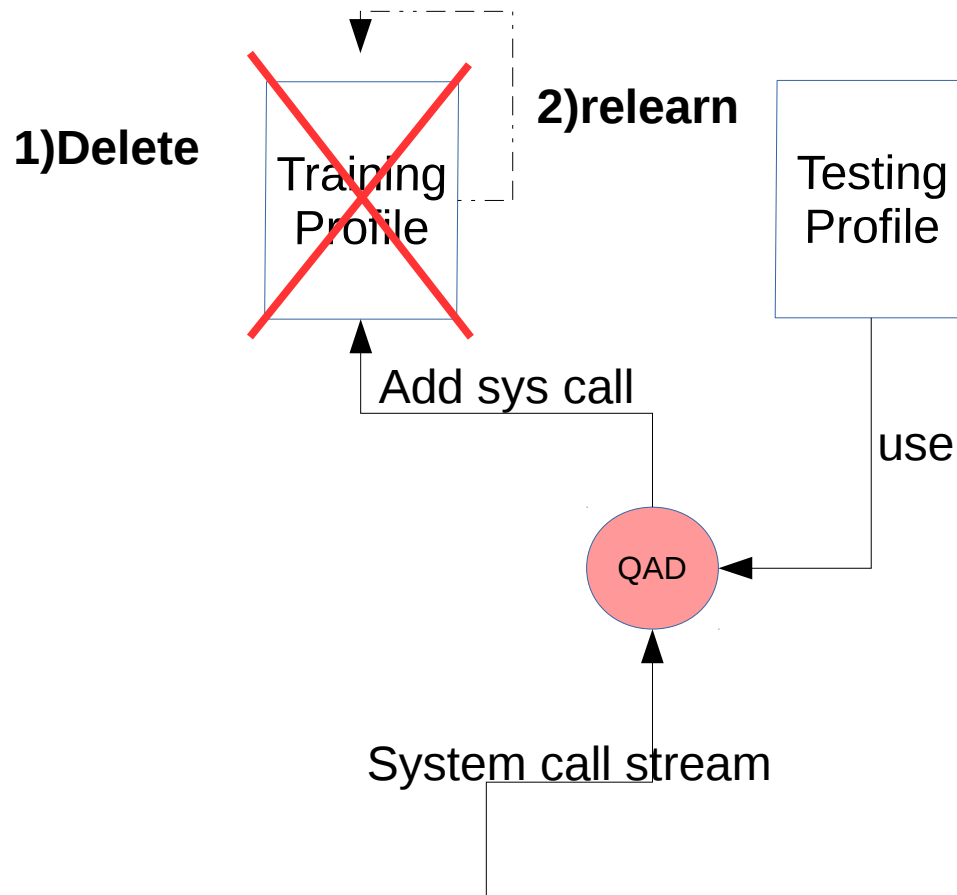
Tolerize

- We tolerize (or accept as normal behavior) if we detect $> \text{anomaly_limit}$ anomalies.
- Testing profile is deleted and a copy of training to testing occurs.



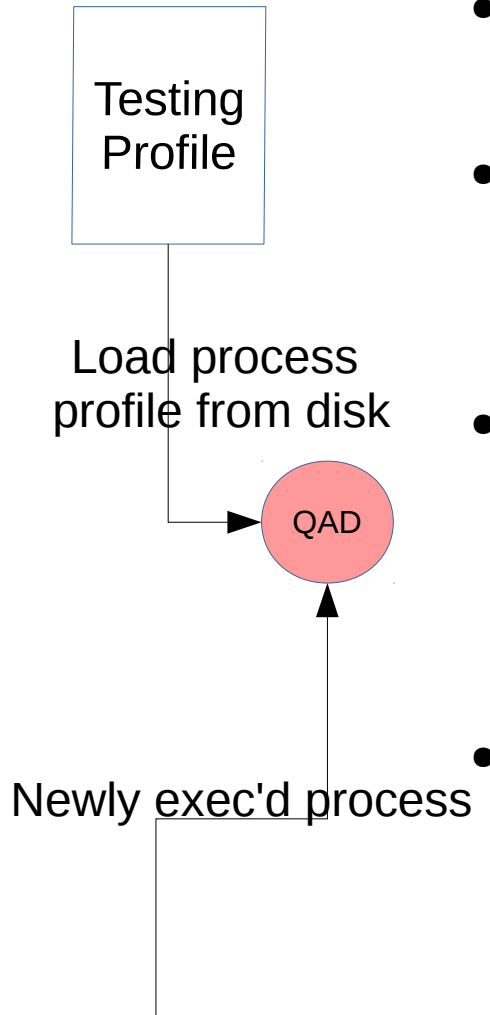
Sensitize

- We sensitize when we're learning bad behavior If $LFC > \text{tolrize_limit}$, we wipe out the training profile and re-learn



On process creation

- Testing profile is loaded from disk
- Multiple execs of the same process points to the same profile
- If a subverted process calls exec, this might trick us. Process creation might be prohibited for a process.



Limitations!

- Mimicry attacks – could be mitigated by using bigger window sizes and including system call arguments)
- Not intended to be an all-inclusive security solution. (Must be used with other security measures)
- We might not train on every possible normal behavior (in my opinion this is indicative of a verification problem)

Limitations!

- Normal is not a well-defined concept. Disk failure, timeout over networks and a successful attack, all would cause this not to behave normally.
- It is hard to exercise all normal behavior modes for any non-trivial program.
- If we're learning “out in the wild” (which I do not suggest doing). Data could be contaminated with attack patterns