Using Observations of Invariant Behavior to Detect Malicious Agency in Distributed Environments

Thomas Richard McEvoy Stephen D. Wolthusen

Department of Mathematics
Royal Holloway, University of London
Egham, Surrey TW20 0EX
United Kingdom
{T.R.McEvoy , Stephen.Wolthusen}@rhul.ac.uk

Abstract: Detecting malicious software used for covert ends is problematical because skilled attackers invariably employ stealth mechanisms to conceal the injection and subsequent activity of such software. As a result, the evidence of such incursions frequently "disappears" once the attack has succeeded. In distributed environments, this difficulty is compounded because of the inherent difficulties in observing the global state of a computation.

We propose a novel approach to the detection of potentially malicious activity in distributed environments. We select key data elements, which are chosen on the basis that they are frequently subject to subversion during malicious attacks. We specify their behavior as a partial order of sequences in state, accounting not only for legal and illegal states, but also for less than normative behavior, whose occurrence may indicate the presence of anomalous conditions.

We show how we overcome the difficulties of observing state in distributed environments through employing a multiplicity of distinct and independent observer processes and by making use of well-known algorithms to synchronize and order our observations and we demonstrate that we are able to use the resulting data set to make inferences about the presence (or not) of malicious software based on comparisons of observed and expected behaviors.

1 Introduction

Malicious software activity used for covert purposes such as the exfiltration of data is difficult to detect because it generally employs stealth and anti-forensics techniques – including direct attacks on detection mechanisms. As a result, evidence of its activity may only be momentarily available either during incursion or on subsequent activation (Section 6). On distributed systems, this problem is compounded because the global state of the system may not be observed directly, due to the computational unfeasibility (Section 2). This problem is an inherent feature of distributed systems and also applies to software engineering issues, such as debugging and deadlock detection [MG01].

We identify – for observation – data elements, which are frequently targeted for subversion during an attack (Section 3). We show how the behavior of these data elements relates to

the behavior of the distributed system as a whole (Sections 4.1, 4.2). We describe the expected behavior of these data elements by modified state diagrams, using knowledge of invariant behaviors in relation to causality, or system security policies (Section 4.3, 4.4). We create from combining these graphs, a "fuzzy" partial order over the combined sequences of these states. From this, we may induce a resulting set of total orders and associated probability values.(Section 4.5).

We implement a multi-pronged observation of these elements using a set of independent and distinct observer processes and utilize well-known distributed algorithms to synchronize these readings, to provide a logical time frame, and – from time to time - record the global state of the observation as a whole (Section 4.6). We derive from this set of observations a second "fuzzy" partial order of observed states and induce a corresponding set of linearizations. Since the predicted and observed sets are equivalent, by comparing these sets, we may uncover anomalous behaviors. (Section 4.7).

The results are partial and probabilistic in nature, but the validity and integrity of the data set is underscored by the multiplicity, distinctness and independence of the observers (Section 4.8). The mechanism is also capable of self protection, since it is itself also a distributed system which possesses key data elements which may be likewise observed as part of the mechanism's own working (Section 4.9).

We list some early results using a multiprocessor system to simulate some of the problems of a distributed environment (Section 5). We discuss related work (Section 6) and conclude with a summary of the advantages of our approach. We intend to explore its application to several types of distributed and parallel processing environments and we list some early results (Section7).

2 Detection of Malicious Behavior in Distributed Systems

A distributed system consists of a set N of processes, which are physically, or logically, remote from one another. Each process executes independently. They do not share memory and they exchange information using messages via agreed communication channels. These messages may be subject to delay and arrive out of order. Processes do not share a clock e.g. for time stamping messages or events [Gar02].

From the point of view of an all-seeing observer with a physical clock, therefore, messages and states associated with a run of a distributed computation would happen in a number of different total orders (or linearizations) – which may or may not be valid. This environment therefore poses a difficulty in determining – for example, for purposes of debugging – the validity of a given computational run i.e. detecting whether a particular condition (referred to as a global predicate) such as an "illegal state", is true, or becomes true during a run of a system. In particular, for unstable global predicates (conditions which become true momentarily, but do not necessarily remain true, unlike a stable condition such as termination), this problem is known to be NP-complete [Gar02]. Malicious software detection falls into this category (Section 6).

We require to show that we can reduce this problem to a computationally feasible one

that allows us to make valid observations and inferences about the global state of the distributed computation. Given that a skillful and knowledgeable attacker will be aware of our detection mechanism, we should also show that our approach is resilient to direct attack upon it.

3 Concurrent Observation of Invariants

We focus our attention on a set X selected data elements, which are commonly attacked by malicious agencies [1]. In addition, we select the data elements of X for the characteristic of invariance - that is, under normal operating conditions, it is possible to specify their behavior for all runs of a distributed computation. For example, they may be values which enforce system security policies, such as privilege flags associated with a user login, they may be measurements of systems activity such as the number of network connections, or they may represent fundamental features of the systems operation such as kernel structures. The selection is such that the features are unlikely to vary in operation over the lifetime of the system, or their behavior is well-managed such that all variations are accounted for during the lifetime of the system. It may, however, be necessary to add data elements to the set where there is a significant change in attack behavior.

We represent this behavior of these data elements as a partially ordered set of transitions in state with associated probability values for successor preference (section 4.3). By modeling two or more elements $x \in X$, we may specify causal relations between the states of these elements. Where the behavior of an element $x \in X$ replicates, these linearizations may be represented as a bounded n-tuple of states. From this predictor set, we can induce a set of all possible linearizations of the states of X.

This set of predicted sequences provides the probability space for transitions in state for these elements. It is also isomorphic to a "fuzzy" set, in that each bounded tuple may be present (as members) in the run of a distributed computation to a greater or lesser degree. Moreover, it not only allows legal and illegal behavior to be distinguished, but also provides a finely grained distinction between legal, but unlikely states, which may be indicative of error conditions, and legal, and reasonably probable, states, indicative of normal operation. It follows that if we are able to observe the states of the selected data elements and place them in an an equivalent structure, we have a basis for determining the existence of anomalous behavior, which may be a sign of malicious software activity.

We propose to observe the behavior of our selected data elements by using a multiplicity of distinct observer processes assigned to each data element(section 4.6). Since our observers also form a distributed system, we face the same constraints on process scheduling and messaging as any such system. But by using well-known distributed computing techniques for synchronizing and ordering the observations and for taking snapshots of state [Gar02], we are able to induce a set of possible partial orders of the states of our elements, along with associated probabilities.

[1] based on a knowledge of evolution in attack behaviors

Using our knowledge of system behavior, we may infer from this a set of possible partial orders with varying degrees of membership with regard to the behavior of the system. We compare bounded sequences from the set of total orders, which can be derived from this, with the sequences of the predictor set, and report any mismatches, including low probability conditions, as indicative of possible malicious activity.

We also outline how this approach provides a basis for self defence of the mechanism by allowing it the ability to observe itself. In addition, the employment of numerous independent processes provides a source of resilience and also acts as a deterrent to attack, since even with full knowledge of our mechanism, the attacker finds himself in a position where he is unable to observe, still less control, the operation of the mechanism. Thus, we manipulate the disadvantages of the distributed environment to our advantage.

4 Model Description

4.1 Modeling a Distributed Computation

A distributed system consists of a set P of N processes, $P = \{P_1, P_2, \ldots, P_N\}$. Each process P_i , where $1 \leq i \leq N$ passes through a finite sequence m of local states $(s_1, s_2, \ldots, s_3)$ where $m \geq 1$.

Let S_i be the sequence of local states in P_i. Then S can be defined as

$$S = \bigcup_i S_i$$

A distributed computation trace can be modeled as a tuple $(S_1, S_2, \ldots, S_N, \rightsquigarrow)$, where $\rightsquigarrow$ indicates a state in S_i logically preceding a state in S_j $(1 \leq i \leq j \leq N)$. This structure forms a decomposed partially ordered set (deposet).

A deposet, or run of a distributed program, defines a partial order (the "happened before" relation $\rightarrow$) on the set of states. There exist many total orders (also known as linearizations or global sequences) of this partial order which are sequences of global states where a global state is a vector of local states [Gar02].

We use the *interleaving assumption* for modeling states and events. That is, we do not model simultaneous behavior, but assume for any two concurrent events, say e and f, that e may follow f or f may follow e.

4.2 Modeling Data States

Let X be a selected set of data elements in a distributed computation. For our purposes, the elements of X are selected on the basis that they behave consistently (i.e. are invariant) under normal operating conditions, but behave arbitrarily under abnormal conditions such

as the incursion of malicious software [2].

Each element $x \in X$ passes through a sequence of states S_x during a run of a distributed computation. Let

$$S_X = \bigcup_{x \in X} S_x$$

It follows that if we are able to induce a partial order over S_X as before, then S_X will form a deposet.

We may model the behavior of P in terms of the behavior of X. That is, since each data element $x \in X$ is associated with a process $P_i \in P$, it follows that there is a monomorphism between S_x and S_i, in that for each state in S_x there are one or more state transitions of S_i. There exists, therefore, a mapping between the global states of S_X and the global states S.

Clearly, the model of behavior derived will be less finely grained than if we had direct access to the actual states of P, but we have the advantage of having a smaller set of behaviors to consider. Moreover, the model proposed allows us to consider behavior more directly relevant to the purposes of intrusion detection.

Thus, we argue that if we record the states of S_X we may feasibly compare the resulting set of global sequences (or parts thereof) with an equivalent set of predicted sequences and use our findings to uncover the presence of malicious software.

4.3 Specifying the Behavior of a Single Data Element

We may specify the legal states and permitted transitions of an $x \in X$, denoted $x_i \rightarrow x_j$, using a modified state diagram, which we call an *expected behavior graph* [3]. Where a choice of possible transitions exists, we may also, based on a knowledge of the system, label the edges to show the probability that a given transition will occur [4]. For convenience, we may choose to represent a replicated state as a new node and indicate this using an asterisk. See Figure 1 for an example.

This graph is useful as it enables us not only to distinguish between legal and illegal behavior, but also to identify low probability behavior which may be indicative of anomalous conditions. In effect, the specification serves not only as a description of the probability space Ω of the behavior of x, but also as a "fuzzy" partial order for the relation "preferred successor state" which expresses what behaviors are more likely that others to belong to a set of normal behaviors for the computation.

Clearly, it is infeasible for the graph to express a complete set of global sequences (or even one), which might occur during an extended run of the computation. Rather we designate

[2] Clearly, the selection criteria may be altered for different applications of the method illustrated in this paper.

[3] We do not use loops as we assume a data element remains constant unless acted on and we make some additional changes which are explained in the text.

[4] Similar to the concept of "likely" and "unlikely" in Linux kernel programs used to deal with error conditions.

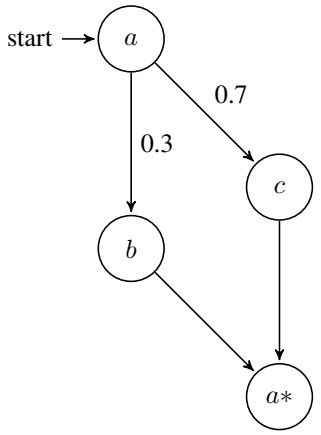

Figure 1: Expected Behavior Graph Showing Legal States and Transitions of x

the initial condition, the *maximum state* (or *maximal*, if it is repeated), any subsequent state which is replicated with its initial conditions, a *sub-maximal state*, and any terminal state as the *minimum state*, or *minimal states* if there is more than one. A set of global sequences may be modeled by chaining these bounded sequences together in various combinations. However, this is not necessary for our purposes.

4.4 Specifying the Behavior of Multiple Elements

We may extend this approach to specify the behavior of more than one data element concurrently, showing each data element as distinct components of the same expected behavior graph. This is meaningful where causal relations exist between the states of different elements. We formally define three such relations – *conditional dependency* , *strong causal dependency* and *weak causal dependency*.

These relations may be used to specify where the relation "potentially causes" ($\rightarrow_p$) must occur in our model, allowing us to induce a partial order over the predicted states of S_X, and hence deduce a set of corresponding global sequences, or rather parts thereof, which is consistent with the "happened before" relation [Gar02].

Where a data element $x \in X$ may only achieve a given state, say x_j, concurrent with a state of another element $y \in X$,say y_k, x_i is said to be conditionally dependent on y_k.

Definition(Conditional Dependency)

Let $x, y \in X$ be data elements. Let $x_i, x_j \in S_x$ be states of x and $y_k \in S_y$ be a state of y respectively. If x_i may not transition to x_j unless y is in y_k, we say that x_j is ***conditionally dependent*** *on y_k.*

We show this relation as a directed edge, which is dashed, on our expected behavior graph from y_k to x_j. y_k is called the *permitting node* of x_j. x_j is called the *dependent node*

of y_k. A joint dependency may also exist, shown by linking co-permitting nodes with an undirected dashed edge. This is called a *permitting component*. Clearly, a permitting node is also a permitting component (see Figure 2).

Two other forms of dependency may also exist where a transition in one component mandates a transition in the other. We graph this by showing a graph with both the "master" and "slave" components and directing an edge from the "master" component to the "slave" component (again, see Figure 2).

Definition(Strong Causal Dependency)

*Let $x_i, x_j \in S_x$ and $y_k \in S_y$ be states of $x, y \in X$. If the transition to y_k forces x_i to transition to x_j, and the transition to x_j only occurs due to y_k, we say that y_k strongly causes x_j, and we call this a **strong causal dependency**.*

The definition for *weak causal dependency* is similar, but removes the condition of uniqueness.

Definition(Weak Causal Dependency)

*Let $x_i, x_j \in S_x$ and $y_k \in S_y$ be states of $x, y \in X$. If the transition to y_k forces x_i to transition to x_j, and the transition to x_j may also occur as a result of other conditions, we say that y_k weakly causes x_j, and we call this a **weak causal dependency**.*

In some cases, a two-way dependency may exist. For example, if two processes are in communication, either process may end the conversation, forcing the other process to do the same.

4.5 Inducing a Partial Order Over Predicted States

Clearly, therefore, we are capable of inducing a partial order over a predicted set of states which when combined will allow us to model the possible global sequences of S_X. We denote this the *predictor set* R_X. This will consist of a set of tuples $(R_x, R_y, ..., \rightsquigarrow_p)$ – where $\rightsquigarrow_p$ indicates the relation (remotely) "potentially causes", which is consistent with the "happened before" relation [Gar02].

For example. Let $(\ldots)$ indicate an ordered, bounded n-tuple of states. Let $[\ldots]$ indicate a permutation of states. Let $\langle(\ldots), (\omega_i)\rangle$ be an ordered pair which indicates the probability of an ordered tuple. Let $\perp$ indicate process termination. Let $*$ indicate a replication of maximal states. Let $X = \{red, blue\}$ be two distinct data elements with causal relations between their respective states.

From Figure 2, the set of expected orders of red and $blue$ [5] are –

$$red = \{\langle(a, b, d, \perp), (0.03)\rangle,$$
$$\langle(a, b, e, a*), (0.27)\rangle,$$
$$\langle(a, c, e, a*), (0.28)\rangle,$$

[5] shown as white and gray respectively

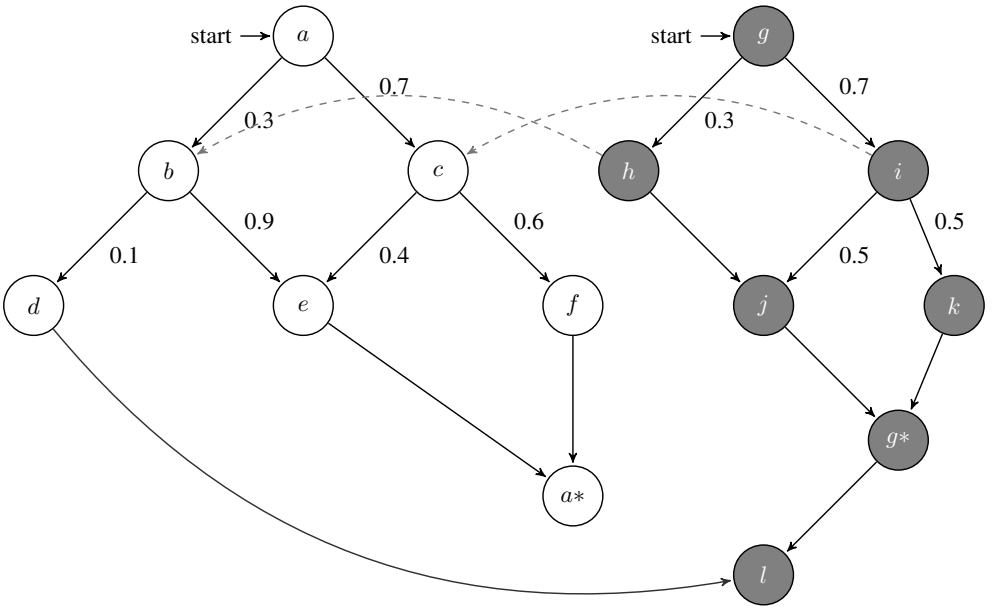

Figure 2: Expected Behavior Graph with Dependencies

$$\langle (a, c, f, a*), (0.42) \rangle \}$$

$$\begin{aligned} blue = \{ & \langle (g, h, j, g*), (0.3) \rangle, \\ & \langle (g, i, j, g*), (0.35) \rangle, \\ & \langle (g, i, k, g*), (0.35) \rangle \} \end{aligned}$$

while the set of partial orders for X, based on the causal relations which exist between these sets of sequences, consists of

$$\begin{aligned} R_X &= \{X_1, X_2, \ldots, X_6\} \\ &= \{ \langle ([ag], h, b, d, l, \perp), (0.03) \rangle, \\ & \langle ([ag], h, b, [ej], *), (0.27) \rangle, \\ & \langle ([ag], i, c, [ej], *), (0.14) \rangle, \\ & \langle ([ag], i, c, [ek], *), (0.14) \rangle, \\ & \langle ([ag], i, c, [fj], *), (0.21) \rangle, \end{aligned}$$

$$\langle([ag], i, c, [fk], *), (0.21)\rangle\}.$$

Taking the element of R_{X_2} in the order shown as an example, using the interleaving assumption, the set of linearized sequences would be

$$R_{X_1} = \{(a, g, h, b, e, j, *), (g, a, h, b, e, j, *), (a, g, h, b, j, e, *), (g, a, h, b, j, e, *)\}$$

,

each of which may occur with $p = 0.675$. The boundary states are the maxmimals a and g and the minimals d and l, respectively. The information provided is sufficient to model a global sequence, if required.

4.6 Observing Data Element Behavior

We show that we can observe the states of X in an order which closely approximates the logical order in which they occurred. To do so, we create a set of observer processes and assign to each data element a partition (or ensemble) of these processes, which takes periodic measurements of its state.

We weakly synchronize our observations by requiring that each observer in an ensemble exchange messages post-observation with every associated observer before proceeding to the next round of observations [Gar02]. This forces the condition that each set of observations must be temporally distinct from each previous round of observations since any process having completed its observation and messaging must wait until it has received from every other process a message concerning their observations before continuing. Hence all observations must have taken place in that round (as there cannot be a message about an observation which has not taken place) before the next round of observations commences.

Messages concerning observations enable us to uncover mismatches in state as the result of each observation is compared by every observer process with its own measurement. We assume that a mismatch indicates a change in state between two distinct observations. This means we avoid the need to log the values from every round of observations and need only log each mismatch in observations.

Observations are not continuous but by using a multiplicity of observers decrease the probability of missing a transition (section 4.8). The order in which transitions are observed and reported within a round for a single data element is not necessarily sequential. However, the order in which transitions are logged across rounds is necessarily sequential for the reasons already discussed in this section.

We use a vector clock [SC02] to logically time stamp messages in accordance with the "happened before" relation [TG98], so that communications regarding observations may be ordered. We use the "pulse" index of a synchronizer algorithm [Gar02] to allow us – if necessary – to identify messages arriving ahead of order to be buffered for future processing. From time to time, we also employ a snapshot algorithm [HS97] to synchronize

the vector clocks between ensembles as part of logically ordering interactions between the states of distinct data elements, using the "happened before" relation.

Hence, for each data element, the results of observation rounds are logged in order, while a partial order of events may be established between the measurement rounds of distinct ensembles, using snapshots in state in combination with vector clock values.

4.7 Inferring Data Element Behavior and the Presence of Malicious Software

The partially ordered set of observed values which we denote T_X is only an approximation of the set S_X which we are able to create from our predictor set R_X for several reasons –

1. Several transitions may take place during a single round of observations, for which we have no basis in observation for determining the order in which they occurred (Section 4.6)
2. There exists a probability we may miss one, or more, transitions in state (Section 4.8)
3. Data may also be missing, or inaccurate, due to the failure of channels or processes, possibly as the result of deliberate action by the attacker, or as the result of spoofing
4. The partial order induced by the "happened before" relation using the vector clock in conjunction with snapshots of state does not map directly to the "potentially causes" relation we identified for R_X

We, therefore, need an approach which deals systematically with these issues and allows us to use the sequences derived from R_X in conjunction with the observed values of T_X to uncover malicious activity.

We deal with reducing the probability of failure, malicious attack and spoofing in section 4.9. In this section, we concentrate on using our knowledge of predicted behavior to make inferences about observed behavior and give some simple examples of how this can be used to uncover evidence of malicious activity. We also demonstrate that our approach is computationally feasible.

To do so, we use both the boundaries we have defined for our sequences and the relations we have identified earlier in creating the predictor set R_X – the "preferred successor" relation, conditional dependency, weak and strong causal dependency – as a lens through which to view the data set T_X. We give examples of various forms of reasoning which may be applied as a result.

Using the maximal, sub-maximal and minimal states we have defined in R_X, we can identify bounded n-tuples of states for single elements. Using the "preferred successor" relation, we may examine the transitions of each element individually. Where a transition is missing, that is, there is no observation of any valid preferred successor, we may infer from R_X a set of probable transitions – which may be presented graphically as a tree

(see Figure 3) – and test these for consistency with subsequent observations, reporting any contradictions, or pruning invalid branches. Additionally, given that the probability of missing transitions will be low, if several appeared to be missing, we would report this as indicative of possible failure in the mechanism, or other anomalous condition.

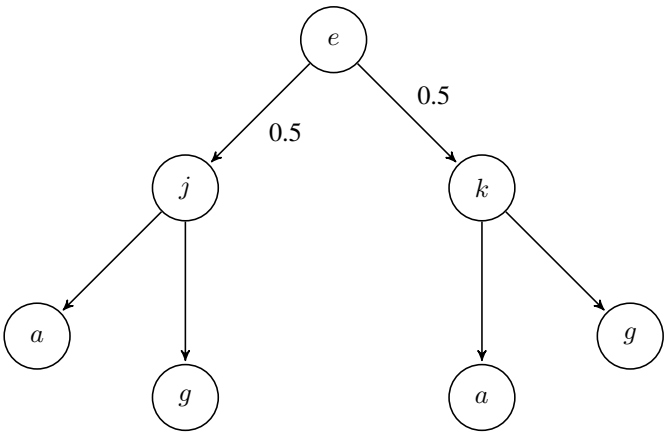

Figure 3: Part of a Hypothesis Tree (for missing observations of j and k) for red and $blue$

Similarly, where we observed several transitions during a single round, we may order these in accordance with the "preferred successor" relation, again presenting the results in a hypothesis tree, checking both that no two transitions lead to contradictory paths and, once again, against subsequent transitions in state for consistency. Once more, it may be possible to prune the number of combinations of transitions as a result of these consistency checks.

When we come to a joint consideration of causally related data elements, we are primarily checking for a consistency in the ordering of these causal relations, that is, that they are congruent with the "happened before" relation which is induced by the vector clock.

Taking strong causal dependency as an example (similar reasoning may be applied to the other relations), we assume that any "slave" state must be preceded by a "master" state . Hence for any two data elements – considering the states of each in order of observation – where we find an initial "slave", we look for an initial "master" state, which may be either observed, or, if missing from the set of observations, inferred – so long as its existence is consistent with the subsequent set of transitions in its associated data element. Similarly, where we find a "master" state, we may look for the "slave" state which is its consequence.

This analysis enables us to associate bounded n-tuples (i.e. sequences) of distinct data elements using "potential causality". We may proceed to check if this relation is consistent with the "happened before" relation over T_X by using vector clock values to link subsets of bounded sequences in a partial order. Where there is an apparent contradiction between the order induced by vector clock values and the order observed or inferred for our "potentially causes" relations, we log this as an anomaly.

In addition, we may also apply the "pigeon-hole" principle i.e. if there are, for example, more "slave" states than "master" states, this is also inconsistent with our predictor set.

It should be noted that, on occasions, establishing the partial order "potentially causes" will enable us to prune back on hypotheses concerning missing data, or multiple transitions in rounds, as the inference of a causal condition or of a permitted, or "slave", state may clarify what transitions have occurred. This is one of the advantages of our approach since it permits us to look at transitions in state from multiple points of view.

So from an initial set T_X of observations, making use of our knowledge of invariant behavior within the system in relation to transitions in state and causal relations between the values of data elements, we are able to infer a "fuzzy" partially ordered set of states of data elements, which we denote $T_X^{'}$, whose sequences and ordering may be checked for consistency against our predictor set R_X and for internal consistency. This gives us a basis for uncovering anomalies which are potentially the result of malicious activity.

As a final step, we may also give consideration to the overall state of the data set in the context of our analysis. We have already stated that low probability – that is, abnormal – conditions may result in an alert and that we might equally raise an alert where several sets of transitions appeared to be missing. In addition, we should also consider where various negative conditions – such as missing data, evidence of failed processes or low probability states - combine and establish a threshold of significance for the quality of the data set. Although this is an approximate concept, we hold that multiple caused failures in the data set may also be considered anomalous and used to trigger an alert.

The computational feasibility of our approach is demonstrated by the fact that we are using well-defined behaviors for our analysis; these behaviors may be considered in the context of bounded, replicating sequences and we can present the results graphically as a tree, therefore, making it possible to avail of well-known graph traversal algorithms to build linear sequences from our observations for analysis and comparison with the set of linear sequences derived from our predictor set R_X.

4.8 Observational Probabilities

There is a probability that one or more states of $x \in X$ may be unobserved. As we have seen, this adds to the complexity of drawing inferences from the data set. At the same time, the use of a multiplicity of observers significantly lowers the probability of this occurring.

Let L be an arbitrary length of time. We observe x using n observers where $n \geq 2$. Let there be a transition in x. We assume this transition δ takes zero time from the point of view of the observers [6] and may happen at any point during L. Let the probability of an observer in the round taking an observation of x before δ be p. Let the probability of taking an observation after δ be q. Let T be the event of observing a transition. If all observations in a round occur before δ or, equivalently, after it, no mismatch in observations occurs. Hence no mismatch in states is reported by the ensemble; this is equivalent to not observing

[6] i.e. either the observation is of state x, or of state x', as any interim state is undefined

it, and we see that $1 - P(T) = p^n + q^n$. Hence,

$$P(T) = \sum_{i=1}^{n-1} ({}^n_i) p^i q^{n-i}$$

This argument may be extended to the observation of further transitions of x during a single round, but it should be clear that where a multiplicity of observers exist, the probability of not observing a transition is much reduced. This reduces the number of possibilities we need to consider when making inferences about our observations.

4.9 Resilience and Self Defence

The multiplicity, independence and distinctness of the observers clearly provides a degree of resilience for the observer mechanism, although performance and messaging overheads need to be taken into account in setting the number of observers. In addition, if poor communications are an issue then some adjustments would need to be made to the synchroniser algorithm to take account of undue latency or communications failure, for example, a controlled time out on message waiting periods.

Considering the possibility of direct attack on the mechanism [Szo05], the major advantage of our approach is that the independence of the observer processes enables us to instantiate a subset of these processes to observe data elements associated with the observation mechanism itself. This enables periodic tests of the integrity of the mechanism. Combined with the possession of a multiplicity of independent observers, this acts as a deterrent to attack by requiring that the capability to simultaneously subvert a set of processes which may not even share the same logical or physical platform, or be functionally equivalent – even if observing the same data element.

A partial subversion of the mechanism is likely to fail because even if a subset of observers are subverted, or have their communications spoofed, this will appear to as a series of continual transitions in state whose permutations are unlikely to be equivalent to the expected behaviors of our selected data elements. If instead we consider an attack *en masse* where the attacker seeks to take advantage of a single vulnerability across an ensemble (or more than one) of observers, we find that this possibility is closed to them. Even within a single ensemble because the same data is observed from different points of view (representing either different APIs or different logical or physical platforms) requiring the use of distinct functionality, the likelihood that a single exploitable vulnerability will result in the mass compromise of the observer processes is low. The use of different platforms also implies that coding or systems operation details may be distinct, further adding to the attacker's difficulties.

Thus, the probabilistic nature of our mechanism acts as a deterrent to attackers since they may not easily predict where and when any actions of theirs may be observed, nor is it computationally feasible for them to seek to control multiple observers which work at different points in process time and space and which possess distinct functionality.

In other words, the constraints which we face in detecting malicious activity are redoubled for the attacker considering the subversion of a distributed computation which is guarded by our mechanism, even with full knowledge of the mechanism. Effectively, we have turned the attacker's apparent strength in being difficult to detect in a distributed environment "jiu jutsu"-like against him[7]

5 Experimental Justification

We summarize findings from an early "proof of concept" of our approach. We analyzed the action of several root kits on a Linux system and identified key invariants in the data structures which were altered by these examples of malicious software in order to subvert the system. A common example were pointers to system functions on the system call table, but more sophisticated attacks targeted other kernel structures such the VFS sub-system, or the Interrupt Descriptor Table (IDT). We used some of these example attacks to build a root kit simulator, which did not perform any malicious actions, but merely sought to conceal file-based information, such as might be disguised by malicious software.

We subsequently designed and implemented a kernel module application, based on a distributed computing architecture, called *KRAKEN* which made use of multiple observer processors to examine our selected data element from different points of view, that is, using different APIs for the purposes of measurement.

The application was based on a symmetric multiprocessing (SMP) system. Observer processes were assigned to different CPUs on instantiation, which not only ensured independence between observational viewpoints, but also introduced a non-deterministic element into process scheduling which we argue, similar to the model proposed here, would make evasion of the action of the anomaly detector problematical. The observations themselves were stateless, but each observer exchanged messages with its peers to compare results and reported any mismatches in state. The mechanism made use of the "synchronizer" algorithm, logical time-values and snapshots in state similar to the approach described in this paper (Section 4.6).

Several experiments were carried out using the root kit simulator to mimic different methods of concealment used by attackers. *KRAKEN* demonstrated particular success ($p \approx 1$) in dealing with root kits when observations were made at both high and low level APIs concurrently and the subversion method relied on concealing the true state of the data element at an interim level as the continuous contradiction between the observed values made detection inevitable. But even where the subversion method attacked a lower level API than any measured, there remained a significant probability of detecting the anomalies, with some dependence on the frequency of measurement (as high as $p \approx 0.6$ under realistic performance constraints).

Although there are some differences between the more tightly bounded environment of a SMP system and larger scale, more loosely structured distributed systems, this early work

[7] If the reader prefers Chinese martial arts, then he should consider the example of Choy Li Fut, where the student is trained to consider all objects in the local environment as weapons in the fight against his opponent.

demonstrated that the use of concurrent and distributed observations of data elements as an approach to ID justifies exploration.

6 Related Work

The difficulty of detecting an unstable global predicate in a distributed system is known to be NP-complete [Gar02]. Malicious activity may be so characterized since, on incursion, the attacker rapidly deploys stealth techniques, including the subversion of ID mechanisms [Szo05, HB05, Skl07, NW06], which conceal signs of its presence. For example, an email may be intercepted and malformed so as to inject malicious software onto a system and – as a first act of such software – the email will be restored to its original state[8].

Considerable research has gone into the observation of both stable and unstable global predicates on distributed systems, primarily motivated by software engineering requirements such as debugging [VD95] [MG01]. This research has led to the identification of various categories of observable predicates. It has established sophisticated and subtle means of identifying temporal, or causal, relations amongst them. This work has been accompanied by the creation of suitable algorithms, which are implemented from within the observed computation [MG95] [CK05] [Ksh96].

We distinguish our approach to observing a distributed environment by proposing a set of independent observers (processes) distinct from the observed system (Section 3) to take measurements of key parts of that system. Using well-known algorithms [HS97] [SC02] [Gar02] to determine the state of these observations and, from there – based on our knowledge of the system – to make inferences about the global state of the system.

Intrusion detection systems may be classified as host-based or network-based [FHL07]. They may use signature-based identification or heuristics [LPR07]. They may depend on statistical analysis for anomaly detection [KG03], or specify acceptable system behaviors, thus identifying unwanted behaviors [Ko00]. There has been an emphasis on switching analysis from examining usage anomalies to process anomaly detection - see [HFS98] for an early example. Our work carries similarities to this approach, but we distinguish it by focusing on the computational outcomes of processes rather than their operation and thus we abstract away from architectural considerations.

We also focus on the identification of causality between independent processes. Although previous work in this area appears to make implicit assumptions [KMLC05] which we do not believe are justified. The timestamps of events or its order are generally assumed to be correct, whereas we recognize that, even on small-scale networks and distributed systems, there are issues with understanding the timing and sequence of events and that too much may be made of the reliability of timestamps [GC06]. We use logical orderings of events to address these issues in line with recognized distributed computing paradigms [Gar02].

Moreover, much ID research focuses on attacks which are large-scale manifestations of malicious activity such as worm infections [KMLC05], rather than low-level covert infor-

[8] private communication

mation gathering – which over the lifetime of a system may be ultimately more damaging [LPKS05] [Emi06]. We believe our approach is uniquely suited to uncovering evidence of the latter, since even momentary misbehaviors in systems may become significant where "invariant" characteristics are violated and it is possible at a higher level of abstraction to link these breaches into a semantically meaningful pattern which may be used for attack identification, or forensics analysis and subsequent development of countermeasures.

7 Conclusion and Future Work

We have shown that by employing a multi-pronged observation mechanism and employing well-known algorithms from distributed computing, we may from a knowledge of the expected and observed states of key data elements, make inferences concerning the presence or absence of malicious software. Although the observation is inherently partial and the inferences may be considered probabilistic and approximate, the multiplicity of the observers underpins the validity of our approach. This last aspect also makes the mechanism difficult of subversion. The approach modeled also has the potential to detect direct attack upon it through self observation.

We consider the probabilistic nature of our mechanism (see Section 4.8) to be a deterrent since, while we assume the attacker has knowledge of our mechanism, such knowledge does not help him as he is not able to either predict or control the process of observation, and hence set himself to pass unobserved.

There are limitations on the currently proposed model. We currently require that the observers exchange a complete set of messages with each other during each round of observation. Clearly, where a large number of observers exist, or where communication is low bandwidth this could cause performance issues. This would need to be addressed through adjusting the synchroniser algorithm. Moreover, similar adjustments would be required to cope with poor communications (possibly due to malicious action) leading to messages being dropped. Nor have we formally addressed the observation of mobile processes, where new processes may be created, processes may terminate and channels similarly be created or destroyed.

The system would also require some mechanism for distinguishing genuine communications from spoofed messages. On high performance systems, this could be achieved using encryption mechanisms, but on low bandwidth or low powered systems this would not be feasible. Another approach is hinted at in section 4.9 where we show that the multiplicity of observers employed provides a basis for detecting contradictory messages and logging these as anomalous. As modern platforms migrate toward increased use of multiple processors on a single base, this justifies our approach.

We summarize some early results from a 'proof of concept' version in a multiprocessor system, which we are currently using to simulate some of the problems of working in parallel and distributed environments, and we intend to extend this work to similar larger-scale environments in future.

References

[CK05] Punit Chandra and Ajay D. Kshemkalyani. Causality-Based Predicate Detection across Space and Time. *IEEE Transactions on Computers*, 54(11):1438–1453, 2005.

[Emi06] Aaron Emigh. The Crimeware Landscape: Malware, Phishing, Identity Theft and Beyond. *Journal of Digital Forensic Practice*, 1:245 – 260, 2006.

[FHL07] Yingfang Fu, Jingsha He, and Guorui Li. A Distributed Intrusion Detection Scheme for Mobile Ad Hoc Networks. *COMPSAC '07: Proceedings of the 31st Annual International Computer Software and Applications Conference - Vol. 2- (COMPSAC 2007)*, 02:75–80, 2007.

[Gar02] Vijay K. Garg. *Elements of Distributed Computing*, chapter 1,2,3,4,10,11,12,20. John Wiley and Sons Inc, 2002.

[GC06] Ashish Gehani and Surendar Chandra. PAST: Probabilistic Authentication of Sensor Timestamps. In *ACSAC '06: Proceedings of the 22nd Annual Computer Security Applications Conference on Annual Computer Security Applications Conference*, pages 439–448, Washington, DC, USA, 2006. IEEE Computer Society.

[HB05] Greg Hoglund and James Butler. *Rootkits*. Addison-Wesley, 2005.

[HFS98] Steven A. Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security, 1998*, 6(3):151–180, 1998.

[HS97] Letian He and Yongqiang Sun. On Distributed Snapshot Algorithms. *APDC '97: Proceedings of the 1997 Advances in Parallel and Distributed Computing Conference (APDC '97)*, page 291, 1997.

[KG03] Oleg Kachirski and Ratan Guha. Effective Intrusion Detection Using Multiple Sensors in Wireless Ad Hoc Networks. In *HICSS '03: Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 2*, page 57.1, Washington, DC, USA, 2003. IEEE Computer Society.

[KMLC05] Samuel T. King, Z. Morley Mao, Dominic G. Lucchetti, and Peter M. Chen. Enriching Intrusion Alerts Through Multi-host Causality. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS)*, 2005.

[Ko00] Calvin Ko. Logic Induction of Valid Behavior Specifications for Intrusion Detection. *SP '00: Proceedings of the 2000 IEEE Symposium on Security and Privacy*, 00:0142, 2000.

[Ksh96] Ajay D. Kshemkalyani. Temporal interactions of intervals in distributed systems. *Journal of Computer and System Sciences*, 52(2):287–298, 1996.

[LPKS05] Michael E. Locasto, Janak J. Parekh, Angelos D. Keromytis, and Salvatore J. Stolfo. Towards Collaborative Security and P2P Intrusion Detection. In *Proceedings of the IEEE Information Assurance Workshop (IAW)*, June 2005.

[LPR07] Adrian P. Lauf, Richard A. Peters, and William H. Robinson. Embedded Intelligent Intrusion Detection: A Behavior-Based Approach. *AINAW '07: Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops*, 1:816–821, 2007.

[MG95] J.R. Mitchell and V.K. Garg. Deriving Distributed Algorithms from a General Predicate Detector. *Computer Software and Applications Conference, 1995. COMPSAC 95. Proceedings., Nineteenth Annual International*, 00:268, 1995.

[MG01] Neeraj Mittal and Vijay K. Garg. On Detecting Global Predicates in Distributed Computations. *21st IEEE International Conference on Distributed Computing Systems (ICDCS'01)*, 00:0003, 2001.

[NW06] Naoyuki Nagatou and Takuo Watanabe. Run-Time Detection of Covert Channels. In *ARES '06: Proceedings of the First International Conference on Availability, Reliability and Security*, pages 577–584, Washington, DC, USA, 2006. IEEE Computer Society.

[SC02] Chengzheng Sun and Wentong Cai. Capturing Causality by Compressed Vector Clock in Real-Time Group Editors. *IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium*, page 89, 2002.

[Skl07] Ivan Sklyarov. *Programming Linux Hacker Tools Uncovered*, chapter 21. A-LIST, LLC, 2007.

[Szo05] Peter Szor. *The Art of Computer Virus Research and Defense*. Addison-Wesley, 2005.

[TG98] Ashis Tarafdar and Vijay K. Garg. Happened Before is the Wrong Model for Potential Causality. Technical Report ECE-PDS-1998-006, Parallel and Distributed Systems Laboratory, ECE Dept. University of Texas at Austin, 1998.

[VD95] S. Venkatesan and Brahma Dathan. Testing and Debugging Distributed Programs Using Global Predicates. *IEEE Transactions on Software Engineering*, 21(2):163–177, 1995.