Applying Formal Evaluation to Worm Defense Design

Raman Sharykin
Department of Computer Science
University of Illinois Urbana-Champaign
201 N Goodwin Avenue
Urbana, IL 61801

Phillip A. Porras
Computer Science Laboratory
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025

Abstract

We discuss the early insertion of formal analyses in distributed malware defense evaluation, and provide an example method for applying an executable rewriting logic specification to drive both simulation and property validation of a collaborative group-based worm defense. An important aspect of the algorithm under consideration is its distributed and probabilistic nature, which makes the defense system harder to attack but unfortunately also complicates the ability of designers to fully understand its behavioral properties. We demonstrate one approach to formally analyze our case study worm defense algorithm, employing tools that facilitate both statistical simulation and property validation. Our approach is posed as complementary to the current practice of informal design specification and evaluation through network simulation.

1 Introduction

With the increasing importance and complexity of distributed malware defense systems, the application of formal methods for understanding the dynamics of such systems early in their design could prove highly valuable. However, to date formal analyses of malware defense algorithms have been extremely limited, with the vast majority of designers relying on informal or pseudocode specifications, network simulation, and functional testing to assess their designs. One obstacle to the application of formal methods in this area is the difficulty in determining a proper formalism to apply when evaluating a given malware defense algorithm. If an underlying formalism such as Petri nets or discrete-time Markov chains is selected, it may be costly to switch to a different formalism in cases where another formalism is later determined to be more appropriate in assessing certain algorithm properties.

We present an approach to applying formal methods to the design of malware defenses using *rewriting logic* [12]. Rewriting logic allows an early insertion of formal methods, while not restricting the designer to a narrow formalism. It has been shown that different mathematical models can be naturally expressed in rewriting logic [11, 12]. Formalisms based on rewriting logic can be concise, relatively intuitive, and well suited for specifying distributed concurrent systems with asynchronous communication in play. Rewriting logic specifications are executable in a rewriting-logic language, such as Maude [8], which allows one to simulate and adjust a malware defense specification over various attack scenarios very early in the design life cycle. When the designer is satisfied with the specification, an appropriate narrower formalism with a known representation in rewriting logic can be used to assist in proving various critical properties or to identify logical inconsistencies not discernible through simulation. Maude itself supports assistance for formal proofs in rewriting logic [7].

Once a formalism is selected, another important question is how to analyze the system during early development. A desirable method should be "lightweight," in that it should allow the designer to focus on the defense algorithm and its parameters. We prefer methods that allow us to rapidly assess key behavioral properties and iterate the defense model under various operating assumptions, and to invest efforts in more complex formal validation procedures only later, when the design space is narrowed. For our application of a malware defense system, we propose a simulation-based approach with a temporal quantitative language QuaTEx [1], which we explain is well suited to quantitatively assessing the behavior of stochastic systems, such as malware defense protocols under a distributed attack.

To illustrate our methodology we consider a case study involving a probabilistic worm defense algorithm. The primary contribution of this paper is to demonstrate a formal approach to analyzing the behavioral properties of this peer-based distributed stochastic system in a malware defense context, and to illustrate the types of security properties that we believe are applicable, here and with other malware defense algorithms, to investigation through formal analyses rather than simulation.

The paper is organized as follows: Section 2 introduces the illustrative worm defense system; Section 3 explains how this system can be specified using Maude rewriting logic; Section 4 explains how a property can be specified in QuaTEx and how it can be statistically analyzed using VeStA; Section 5 discusses example properties and introduces a property which cannot be inferred from analysis of a propagation curve; finally, Section 6 presents our findings while shaping our algorithm and provides technical results.

2 An Example Worm Defense System

To motivate our presentation, we briefly present an example, previously published, group-based worm defense algorithm [4]. Our intent here is to describe the basics of the algorithm to drive our formal modeling discussion, and refer the reader to the publication for more information regarding the algorithm details and efficacy arguments.

Under this group defense algorithm, local area networks (LANs) collaborate in groups, where each LAN's egress router informs its group partners when it produces local alerts associated with potential worm infection activity. When a LAN receives corroborating worm reports from `N` or more sources, it enters a defensive filtering posture. The algorithm is similar to the peer alert sharing protocols presented in [2] and [13], but here alert production is driven by a connection rate-limiting system, such as that presented in [16]. Our connection rate-limiting component produces an alarm when it observes an internal host that attempts to connect to more than a threshold number of unique IP addresses per unit time. Connections to new hosts that exceed the threshold are dropped at the egress router until the next time interval. One benefit of this combined defense strategy observed in closed-network simulations is that while peer-sharing algorithms are fundamentally subject to defeat by rapid worm propagation, the connection rate-limiting system that produces the alerts also effectively slows overall worm propagation speed well enough to ensure that peer corroboration takes effect.

Upon receiving a sufficient number of corroborating worm infection alarms from itself or peers, an egress router has the capacity to switch into a *defensive posture* (a filtering mode that drops all packets that are assumed correlated with the majority of peer alerts). To arrive at a sufficiently abstract model, we do not specify how to accomplish filtering, but we attribute a cost to filtering and preclude the defense algorithm from simply staying in the defensive posture. Other researchers are developing automatic signature generation systems such as EarlyBird [15] and Autograph [9], which could be used for filtering in an implementation of this scheme.

We model a LAN as a graph of local host nodes with one egress node, and multiple LAN are interconnected to each other via their egress nodes. The overall group defense scheme is modeled as the parallel, asynchronous, deployment of a LAN defense algorithm that is embedded in each egress node. Each LAN defense algorithm instantiation independently progresses through several potential phases at each time interval: local worm detection, peer-group formulation, alert publication, and security posture updating. During the detection phase, the egress node may observe local rate-limit violations at some end nodes, generating a local alert for each violation. During group formulation, the egress node produces a group set of size `G` from the set of participating peer LANs. The group-forming algorithm is designed to ensure a fair distribution of alerts among the collaborating population. At each time interval, an egress node forwards an alert to its peer group if it has produced at least one local worm alert during the current time interval. The egress node also receives alerts from other peer LANs and increments a current local alert level metric `a`, based on the number of local and remote alerts received. Each alert increments `a` by a parameterized `severity` value. The `severity` metric can be adjusted depending on how much or how little corroboration the LAN is required to establish before it will enter a defensive posture. `a` is decayed by a parameterized value per subsequent time intervals. When `a` exceeds the threshold value α,the egress node imposes filtering on all incoming packets that match the filtering criteria. α is calculated as `severity` $\times\ r$, where r indicates the amount of corroboration required before enabling filters. This filtering posture is maintained for the number of time intervals required to enable the decay function to bring `a` back to zero.

3 Worm Defense Specification in Rewriting Logic

We now outline some key Maude features relevant for specifying the above collaborative defense system. A distributed system configuration is modeled in Maude as a collection of concurrent objects and messages that behave according to a set of rewrite rules describing the behavior of individual objects. Maude allows the declaration of node objects with the following syntax:

```
[ identifier | attr_1 : Type_1, ...,
               attr_N : Type_N ]
```

where identifier is a natural number, `attr_1`, . . ., `attr_N` are attribute names of types `Type_1`, . . ., `Type_N`. An example node in the node object set looks like

```
[ 7 | infected : true, alerts : 5,
      alertLevel : 2 ]
```

This represents an example node with identifier 7, having three attributes: `infected`, `alerts`, and `alertLevel` with the corresponding values. In the actual implementation we have several other necessary attributes.

Synchronous and asynchronous communication has a natural representation in rewriting logic. For simplicity we use synchronous communication to model the worm propagation process. A rewrite rule for such a propagation involves two nodes and has the following syntax:

```
rl[O  |infecting: true, infected: true,
       filtering: false, attrSet ]
  [O' |infecting: true, infected: false,
       filtering: false, attrSet ]
  =>
  [O  |infecting: true, infected: true,
       filtering : false, attrSet ]
  [O' |infecting: true, infected: true,
       filtering: false, attrSet ]
```

This rule specifies the infection event of the node with the identifier `O'` by the infected node with the identifier `O`, neither of which filtering. More precise infection models can be specified by using asynchronous communication and introducing time and messages with arrival times in the system.

Another important aspect of the rewriting logic formalism is its probabilistic variant. A *probabilistic rewrite rule* [1, 10] and a non-probabilistic rewrite rule together can be used to specify the propagation action of a random search worm:

```
prl[O |infecting: true, infected: true,
       infect: O'', attrSet ]
   =>
   [O |infecting: true, infected: true,
       infect: O', attrSet] with
       probability O':=uniformDist(IDSet)

rl[O  |infecting: true, infected: true,
       infect: O', attrSet ]
  [O' |infected: false, attrSet' ]
  =>
  [O  |infecting: true, infected: true,
       infect: O', attrSet ]
  [O' |infected: true, attrSet' ]
```

In the first rule, the identifier of the node to be infected is chosen randomly and uniformly from the set `IDSet` of all possible node identifiers. The construction `prl` is a part of the PMaude specification language which is a probabilistic extension of Maude. The formal semantics of PMaude has been described in [1].

The defense system goes through four main phases at each clock cycle: alert production, peer-group formulation, alert publication, and security posture management (i.e., deciding whether incoming alert notifications warrant the enabling of egress filters). We now explain how each phase of our algorithm is specified in rewriting logic. During the detection phase, the egress node's connection rate limiter detects violations of its threshold with a certain probability. An example rewrite rule, in which `detectionProbability` is assumed to be known, is expressed as follows:

```
prl[O |infected : true, detected: X,
       attrSet ]
   =>
   [O |infected : true, detected: Y,
       attrSet ] with probability
       Y:=bernulli(detectionProbability)
```

The next phase is the group forming phase. During this phase each node forms its group randomly. The rewrite rule to form a group may look like

```
cprl[O |group: L, attrSet ]
    =>
    [O |group: O';L, attrSet ]
      with probability O':=uniform(IDSet)
      if size(L) < F
```

where `L` is the list of group member identifiers separated by semicolons, and `cprl` is a *conditional probabilistic rewrite rule*. The rule is applied repeatedly until the group size has reached `F`.

The third phase is alert publication, in which each egress node that detected a local rate-limit violation distributes an alert to the members of its selected peer group. An example rewrite rule specifying this process is

```
rl[O  |infected: true, detected: true,
       group: O';L ]
  [O' |alerts: alertNum, attrSet ]
  =>
  [O  |infected: true, detected: true,
       group: L ]
  [O' |alerts: alertNum + 1, attrSet ]
```

where `O';L` is the list of identifiers of the alert group of the node `O`. The rule is applied repeatedly until all identifiers are consumed by the rule application, increasing the alert levels of those peer nodes in the group.

The last phase for the egress node is that of security posture management. Here, each egress node must independently decide whether to enter or exit the defensive posture based on its current alert level. An example rewrite rule to capture this logic is

```
rl[O |alert: alertLevel,
      filtering: currentStatus, attrSet ]
  =>
  [O |filtering:
      if alertLevel == alpha
         then true
         else if alertLevel == 0
                 then false
                 else currentStatus
              fi
      fi, attrSet ]
```

where `alpha` is the alert threshold for nodes to enter the defensive posture.

$$
\begin{aligned}
&QuaTEx : percentInfected(percentage, count) = \\
&\qquad \underline{if}\ count = 0\ \underline{then}\ percentInfectedInState() \\
&\qquad\qquad \underline{else}\ \underline{if}\ percentInfectedInState() > percentage \\
&\qquad\qquad\qquad \underline{then}\ \bigcirc(percentInfected(percentInfectedInState(), timeSpan)) \\
&\qquad\qquad\qquad \underline{else}\ \bigcirc(percentInfected(n, count - 1)) \\
&Querry : E[percentInfected(0, timeSpan)]
\end{aligned}
$$

Figure 1. Example Quatex Expression

4 Statistical Analysis

Among its advantages, a Maude rewriting specification is also an executable logical specification, allowing us to reason about our egress node logic in distributed, concurrent, and asynchronously communicating network deployment scenarios [11]. To aid our evaluation of the group-collaborative defense logic, Maude provides facilities to support this evaluation by allowing us to specify desirable program properties, and a mechanism to assist in their verification. Here, we present the QuaTEx [1] language as our method for specifying several desired algorithm properties, and statistical model checking supported by the VeStA tool [14] to help us validate these properties.

4.1 QuaTEx

The most commonly known way to state properties over paths in stochastic systems is with probabilistic temporal logics (PTLs). However, PTLs are somewhat restrictive: they are limited to true or false evaluations for a given path of the system, whereas one might want to quantify and compare various path traversal results. For this reason we use the QuaTEx language [1]; the name stands for *Quantitative Temporal Expressions*. The language is supported by the VeStA tool [14], which has an interface to PMaude and enables one to model check PMaude specifications against QuaTEx properties.

The primary objective of QuaTEx is to generalize probabilistic temporal logic formulas from Boolean-valued expressions to real-valued expressions. The Boolean interpretation is preserved as a special case using the real numbers 0 and 1. As usual, QuaTEx has *state expressions* that are evaluated on states, and (real-valued) *path expressions* that are evaluated on computation paths. The notion of state predicates is now generalized to that of *state functions*, which can evaluate quantitative properties of a state. QuaTEx is particularly expressive, and includes an ability to define recursive expressions. In this way, only the next operator $\bigcirc$ and conditional branching *if Bexp then Pexp else Pexp′ fi*, with *Bexp* Boolean and *Pexp*,*Pexp′* path expressions are needed to define more complex operators, such as the until $\mathcal{U}$ of probabilistic computational tree logic (PCTL) and of continuous stochastic logic (CSL) [3], and the CSL-bounded until $\mathcal{U}^{\leq T}$. More details regarding QuaTEx and its semantics can be found in [1]. Figure 1 illustrates one of the QuaTEx expressions evaluated for our case study. This expression is evaluated on computation paths, and captures the number of infected nodes at the time when the number of infected nodes has stabilized. If this formula, $percentInfectedInState()$ is the state function that maps the current Maude state to the percentage of infected nodes at the current time tick.

When translated to the VeStA syntax, the QuaTEx query above instructs VeStA to compute the mathematical expectation of the number of infected nodes after the worm reaches its full saturation. The typical shape of the worm growth dynamics suggests that at this point in time, the system reaches an *equilibrium*. We define the equilibrium as the point in time at which the number of infected nodes has not changed for a $timeSpan$ number of ticks. The recursive-over-time function $percentInfected(percentage, count)$ provides the percentage of infected nodes at the end of the simulation. The simulation ends when there have been no new nodes infected for the predefined time $timeSpan$.

4.2 VeStA

VeStA is a tool that performs *statistical analysis* on a probabilistic system by evaluating QuaTEx expressions on computation paths obtained by Monte Carlo simulations. When two parameters α and δ are provided to the tool, VeStA responds with a real number v, which is the estimated value of the expression with a $(1 - \alpha)100\%$ confidence interval bounded by δ. Depending on the tightness of the parameters, VeStA may need a greater or smaller number of sample runs to compute such a value. An example output of the command above has the following form:

```
Sample count = 1885
Result: 0.2885941
Run time: 6291.132 seconds
```

It shows the number of paths VeStA needed to obtain the result with the confidence interval, the result itself, and the running time in seconds.

5 Design Goals of a Worm Defense

The evaluation and comparison of the emerging number of worm defense approaches remains a research challenge. In a discussion of the need for formalizing evaluation and

$$
\begin{aligned}
Quatex:\ & maxSpeed(speed, percentage, count) = \\
& \quad \underline{if}\ count = 0\ \underline{then}\ speed \\
& \quad\quad \underline{else}\ \underline{if} percentInfectedInState() - percentage > speed \\
& \quad\quad\quad \underline{then}\ \bigcirc\ maxSpeed(percentInfectedInState() - percentage, \\
& \quad\quad\quad\quad\quad\quad\quad\quad percentInfectedInState(), timeSpan) \\
& \quad\quad\quad \underline{else}\ \bigcirc\ maxSpeed(speed, percentage, count - 1) \\
Querry:\ & E[maxSpeed(0, 0, timeSpan)]
\end{aligned}
$$

Figure 2. Maximal Worm Propagation Speed

comparison criteria, [6] observes that today the evaluation of worm defense strategies centers nearly exclusively on the impact that the defense has on the infection growth rate. However, in many cases this is a less than distinguishing metric as many schemes show similar dynamics. Here, we evaluate classical worm defense properties such as the infection growth rate, but also augment our evaluation with properties inspired by [6]. All properties discussed next are stated in QuaTEx, along with text statement of the property. We do not present the infection growth rate graph itself, but rather estimate the three key parameters of its curve. We present two types of properties: (i) values that are possible to infer from infection growth curves, and (ii) values that *cannot* be inferred from infection growth curves.

5.1 Properties Based on Infection Growth Curves

We consider a particular type of worm propagation curve, which occurs in the simulations of our defense algorithm and our worm model. The shape of the curve is sigmoidal (S-curve), with rapid growth followed by the epidemic reaching its equilibrium. Three values characterize curves of this type visually: the total infection percentage at saturation, the maximal propagation speed, and the time at which saturation is reached.

Property 1 *Estimated number of infected hosts after the worm has reached its full saturation.*

The QuaTEx query for Property 1 was presented in the previous section, Figure 1.

Property 2 *The maximal worm propagation speed measured as a percentage of nodes per tick infected during the greatest infection spike is expressed in Figure 2.*

Property 3 *The expectation of the time point when the worm has reached its full saturation is expressed in Figure 3.*

5.2 A Property Independent from Infection Growth Curves

Most contemporary worm defense design assessments concentrate on infection growth impact, which provides a direct insight into the overall protection effectiveness of the defense, but does not capture issues such as the cost associated with defensive filtering, overhead of communications, or local impact to infection resistance for a participating or nonparticipating LAN. We propose a property inspired by [6] and generalized in QuaTEx spirit:

Property 4 *Estimated percentage of uninfected nodes in the nondefensive posture after the worm has reached saturation is expressed in Figure 4.*

6 Analysis of Properties and Discussion

When analyzing Property 1, we observed that gaining high confidence in the analysis requires a large number of runs. This means that the distribution of the random variable under consideration has a large deviation. From this we can infer that the number of infected hosts may vary significantly from run to run. For each run, our system prints out the initial random seed used and the results obtained. This permits us to reconstruct problematic runs by using the same initial random seed for the forensic analysis. In doing so we discovered that since each node chooses its peers at random from the whole population, in some cases the population is not covered uniformly. Thus, some nodes do not get enough alerts to reliably enter the defensive posture in time. That is, uneven coverage of the network leads to the concern that if all nodes happen to cover only a particular part of the network, then the excluded nodes do not get enough alerts to enter the defensive posture in time. The probability of this situation is not high, but it is also not very difficult to avoid the problem.

In light of this finding, we adjusted our group selection mechanism to ensure uniform fair coverage of LANs in groups. In our solution, at every tick one node with low probability becomes a leader. The leader assigns alert groups to the rest of the population such that the coverage is highly uniform. The proposed approach has two benefits: the leader cannot be located ahead of time, and the burden of assigning groups is uniformly distributed throughout the network. The importance of refreshing one's alert groups is demonstrated in [5], which discovered multiple counter-quarantine worm propagation strategies that exploit static group structures.

$$
\begin{aligned}
Quatex:\ & satTime(percentage, count) = \\
& \quad \underline{if}\ count = 0\ \underline{then}\ time() \\
& \quad\quad \underline{else}\ \underline{if}\ percentInfectedInState() > percentage \\
& \quad\quad\quad \underline{then}\ \bigcirc\ satTime(percentInfectedInState(), timeSpan) \\
& \quad\quad\quad \underline{else}\ \bigcirc\ satTime(percentage, count - 1) \\
Querry:\ & E[satTime(0, timeSpan)]
\end{aligned}
$$

Figure 3. Time to Full Saturation

After fixing the unevenness in the population coverage we tried to vary the false negative rate. We found that the number of infected nodes is very sensitive to this parameter when the whole saturation occurs. The number of infected nodes rapidly grew when we stated the false negative rate higher than 5 percent. This can be explained by the fact that this particular algorithm is very sensitive to the time when infected nodes enter the defensive posture, where early entry can significantly hinder an emerging worm. However, this may be unrealistic in general, illustrating that simulations may point to potentially important parameters of the system, and suggesting a need for more study of this phenomenon.

6.1 Experimental Setting

We used VeStA to query properties described in the previous sections. The parameter α was set to 0.01 and the parameter δ was set to 0.05. The choice of α means that if the computation of values is repeated, they will be in the same confidence interval with probability 99%. Given an output v of the algorithm, the confidence interval is computed as $[v(1-\delta), v/(1-\delta)]$. The worm model was a random search worm, which tries to infect a machine at each tick. These parameters are summarized in the following table:

Name	Meaning	Value
t_S	Infection plateau time to declare saturation	10
α	Confidence level	99%
δ	Confidence interval	0.05

The parameters of the defense system include the number of egress nodes, group size, the severity and corroboration metrics, and $alpha$, and were explained in Section 2. These parameters, along with the modeled false positive and negative alert rates per tick (we employed a Bernoulli distribution), are summarized in the following table:

Name	Meaning	Value
N	number of nodes	10
G	group size	4
s	severity	3
r	corroboration	2
$\alpha_t = s * r$	alert threshold	6
p_{fp}	false positive alert rate	0.1
p_{fn}	false negative alert rate	0.05

The worm saturation was defined as the infection percentage stability during 10 ticks. Our model simulations and property validations were conducted on a Linux workstation, with two Pentium 4 Xeon processors and 16Gbs of memory.

6.2 Technical Results

We obtained the following results when computing the proposed properties using the experimental setting described in the previous section. The results were obtained using a version of the algorithm with heuristics for uniform peer selection described earlier.

- **Property 1.** The percentage of infected hosts after saturation lies in the confidence interval $[0.27, 0.30]$. The computation took about 100 minutes, requiring the generation of about 1800 paths.

- **Property 2.** The propagation speed of the worm during its maximum infection spike was measured as a percentage of newly infected nodes per tick, with the confidence interval $[0.14, 0.15]$. The computation of the property took about 20 minutes, requiring the generation of about 300 paths.

- **Property 3.** The expected time to saturation lies in the confidence interval $[14.0, 15.5]$. The computation of the confidence interval took about 16 minutes and required the generation of about 200 paths.

- **Property 4.** The percentage of uninfected nonfiltering hosts lies in the confidence interval $[0.53, 0.58]$. The computation took about 50 minutes and required the generation of about 800 paths.

6.3 Scalability

To address the scalability issue we looked at the dependence of Property 3 to the number of nodes in the system when keeping the ratio (group size / number of nodes) constant. The results show very low dependence of the saturation time on the number of nodes, and are summarized as follows:

$$
\begin{aligned}
Quatex:\ & notInfectedNonFiltering(percentage, count) = \\
& \quad \underline{if}\ count = 0\ \underline{then}\ NotInfectedNonFilteringInState() \\
& \quad\quad \underline{else}\ \underline{if}\ percentInfectedInState() > percentage \\
& \quad\quad\quad \underline{then}\ \bigcirc\ notInfectedNonFiltering(percentInfectedInState(), timeSpan) \\
& \quad\quad\quad \underline{else}\ \bigcirc\ notInfectedNonFiltering(percentage, count - 1) \\
Querry:\ & E[satTime(0, timeSpan)]
\end{aligned}
$$

Figure 4. Estimated Saved Population

Number of Nodes, N	10	50	70	100
Group size, G	4	20	28	40
Saturation time	14.73	15.08	15.34	16.65
Computation time (hours)	0.31	4.41	6.83	18
Paths generated	240	2125	2625	4230

7 Conclusion

We present a formalism and evaluation procedure for examining the behavioral properties of distributed malware defense algorithms. While we have demonstrated our approach using an example collaborative peer-based worm defense, we believe that our method and proposed properties are highly applicable to a number of published malware defense schemes. We suggest an iterative evaluation procedure consisting of three key steps: design specification, property statement, and property analysis. As the formal basis for the first one we have demonstrated the use of rewriting logic; for the second process we employed QuaTEx; and for the third process we employed VeStA together with Maude. We have presented an example specification, four example evaluation properties, the numerical results obtained through our process, and our general experiences in using this approach.

Acknowledgement. This material is based upon work partially supported through the U.S. Army Research Office under the Cyber-TA Research Grant No.W911NF-06-1-0316, by the National Science Foundation under Grants No.ANI-0335299, CNS 05-24516, ONR N00014-02-1-0715, and through a subcontract with the University of California at Davis, Contract No. 01RA005. We cordially thank Linda Briesemeister for working with us on this project.

References

[1] Gul Agha, José Meseguer, and Koushik Sen. PMaude: Rewrite-based specification language for probabilistic object systems. In *3rd Workshop on Quantitative Aspects of Programming Languages (QALP'05) ENTCS, http://osl.cs.uiuc.edu/∼ksen/publications.html*, 2005.

[2] K. G. Anagnostakis, M. B. Greenwald, S. Ioannidis, A. D. Keromytis, and D. Li. A cooperative immunization system for an untrusting Internet. In *Proceedings of the 11th IEEE International Conference on Networks (ICON'03)*, October 2003.

[3] Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert Brayton. Model-checking continuous-time markov chains. *ACM Trans. Comput. Logic*, 1(1), 2000.

[4] Linda Briesemeister and Phillip Porras. Microscopic simulation of a group defense strategy. In *Proceedings of Workshop on Principles of Advanced and Distributed Simulation (PADS)*, June 2005.

[5] Linda Briesemeister and Phillip Porras. Automatically deducing propagation sequences that circumvent a collaborative worm defense. In *Proceedings of the 25th International Performance Computing and Communications Conference (Workshop on Malware)*, April 2006.

[6] Linda Briesemeister and Phillip Porras. Formally specifying design goals of worm defense strategies. In *Proceedings of DETER Community Workshop on Cyber Security Experimentation and Test*, June 2006. Extended Abstract.

[7] M. Clavel, F. Durán, S. Eker, and J. Meseguer. Building equational proving tools by reflection in rewriting logic. In *In Proceedings of the CafeOBJ Symposium '98*. Japan Advanced Institute for Science and Technology, 1998.

[8] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and José Quesada. Maude: specification and programming in rewriting logic. *Theoretical Computer Science*, 285, 2002.

[9] Hyang-Ah Kim and Brad Karp. Autograph: Toward automated, distributed worm signature detection. In *USENIX Security Symposium*, 2004.

[10] Nirman Kumar, Koushik Sen, José Meseguer, and Gul Agha. A rewriting based model of probabilistic distributed object systems, 2003.

[11] Narciso Martí-Oliet and José Meseguer. Rewriting logic: roadmap and bibliography. *Theoretical Computer Science*, 285, 2002.

[12] José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1), 1992.

[13] D. Nojiri, J. Rowe, and K. Levitt. Cooperative response strategies for large scale attack mitigation. In *DARPA Information Survivability Conference and Exposition*, 2003.

[14] Koushik Sen, Mahesh Viswanathan, and Gul Agha. On statistical model checking of stochastic systems. In *CAV*, 2005.

[15] Sumeet Singh, Cristian Estan, George Varghese, and Stefan Savage. Automated worm fingerprinting. In *OSDI*, 2004.

[16] Stuart Staniford. Containment of scanning worms in enterprise networks. *Journal of Computer Security*, 2003.