Technical Report • 11 October 2005

Model Checking of Worm Quarantine and Counter-Quarantine under a Group Defense

Technical Report Number: SRI-CSL-05-03
SRI Project 13738

Prepared by:
Linda Briesemeister, Phillip A. Porras, Ashish Tiwari
Computer Science Laboratory

Approved:
Patrick Lincoln, Director
Computer Science Laboratory

William S. Mark, Vice President
Information and Computing Sciences

Model Checking of Worm Quarantine and Counter-Quarantine under a Group Defense

Technical Report SRI-CSL-05-03

Linda Briesemeister, Phillip A. Porras, Ashish Tiwari
firstname.lastname@sri.com
Computer Science Laboratory
SRI International

October 11, 2005

Abstract

We consider what it means to perform worm quarantine across a network with an emerging self-propagating worm outbreak. It is generally understood that an effective quarantine defense can under certain conditions reduce the infection growth rate, and ideally can prevent a worm from reaching its full saturation potential. This report attempts to more precisely define the desired properties of a quarantine algorithm, and suggest different forms of quarantine properties that vary in their ability to isolate infected nodes, ensure the existence of an uninfected population, and guarantee some persistent protection, no matter how the worm behaves. We employ the SAL formal modeling language and model checker to investigate these properties on a specific group-based quarantine algorithm. In addition to answering questions regarding algorithm correctness and validating some quarantine properties, the model checker disproves other quarantine properties. The proofs and counter-examples produced during this process help in algorithm design and may be useful in informing simulation experiments or building test cases. Using a game theoretic approach, counter-examples of a win scenario for the defense yield insight into smart worm behavior that defeats a known quarantine defense.

Contents

1 Introduction

Over the last decade, large-scale malicious code epidemics have evolved from rare nuisance applications and research curiosities into well-recognized information-based global security threats. The field of worm countermeasure development is active, with several new and derivative strategies being proposed yearly. Recent countermeasure examples include techniques to throttle infection propagation [Staar, Wil02, WWS+04] or hinder the discovery of infection targets [AR05, GKSS03, Pro04], all of which in an effort to slow the progression of an epidemic. Alternatively, worm quarantine applications seek to dynamically isolate the infected population (e.g., through content filtering, address blacklisting, connection blocking) from the population of uninfected systems [AGI+03, NRL03].

We present a formal model of a nontrivial group-based worm quarantine algorithm using the Symbolic Analysis Laboratory (SAL) language [SAL03] and model checker [dMOR+04]. In previous work, we applied our worm quarantine algorithm to a series of simulation experiments to demonstrate the algorithm's ability to prevent random-scan worms of various speeds from reaching their full saturating potential [BP05]. However, simulation is an inherently insufficient methodology for asserting algorithm behavior beyond the scope of the simulated time window. Neither is simulation useful for fully exploring the infection-sequence space of a nondeterministic worm, or for searching for infection sequences that defeat our quarantine strategy.

Using the SAL language we formally articulate both desirable and unwanted properties of our worm quarantine algorithm. We then employ the SAL model checker to examine these properties through the execution of the full state space of our modeled quarantine algorithm as it competes concurrently against a nondeterministic model of a scan-based worm. We use the model checker to examine the conditions under which our algorithm guarantees the eventual segregation of the entire infected population from the uninfected population. We further examine the conditions under which we can guarantee that, upon completion of the quarantine, there remains a nonempty population of uninfected nodes. SAL allows us to examine the conditions under which our algorithm succeeds or fails to deliver benefit, and identifies counter-examples of infection sequences that defeat our desired properties.

These counter-examples that arise in our failed proofs are of particular significance. A counter-example is effectively a winning infection sequence that defeats our definition of successful quarantine. We speculate on the potential to create "smart worms," which are specifically designed to generate counter-quarantine infection sequences against which the given quarantine algorithm cannot defend. We examine a few such counter-example infection sequences and discuss algorithm alterations that address the discovered shortcomings and lead to an improved quarantine strategy.

In the next sections, we introduce the SAL language and present the formal model of our quarantine algorithm, which represents a modular composition of both the internal egress router logic and the group sharing protocol that compose our group defense capability. We discuss our parameter selection, our network model, and our nondeterministic scan-based worm model. We then present definitions of quarantine as properties of our formal model, and employ the SAL model checker to examine the conditions under which we can prove these properties. Finally, we apply model checking to a win situation for the defense. Counter-examples then show the existence of infection sequences that can defeat a specific quarantine algorithm.

2 Preliminaries of the Symbolic Analysis Laboratory (SAL)

We present a very succinct introduction to the SAL specification language [SAL03] that is sufficient for purposes of this study. We use SAL to both define the formal model of our quarantine algorithm as a state transition system, and employ the SAL model checker [dMOR+04] to analyze properties of this system in the presence of scan-based worms.

In the SAL language, a system is described as a *module*. A module can be a *base module* that directly specifies a state machine, or it can represent a composition of base modules. A *base module* consists of input, local, and output variables. These variables define the state space of the module. The initial values or ranges of the local and output variables are defined in an initialization section.

The transitions of the system are specified using guarded commands. Each guarded command consists of a *guard* g and a sequence of assignments to the local and output variables. The guard g is a boolean formula over the state variables. A guarded command represents transitions of the system from a state in which the guard is true to a state obtained by updating the values of state variables to the values given by the assignments. For example, the guarded command

```
x > 0 AND infected --> x' = x - 1; c' = c - 1
```

indicates that if the value of x is positive and the value of boolean variable `infected` is `TRUE`, then we can decrement the values of x and c by one.

Each state variable, say x, in a SAL module can appear in one of two forms in the SAL specification—x or x'. The primed version denotes the "updated" value of x, that is, the value of x in "the next step." In SAL, primed variables can appear in the guard of a transition. These are used to create *zero-delay* modules, that is, modules that respond instantaneously to their inputs.

The behavior of a SAL module is represented by the set of its state sequences, starting with an initial state and related by the transition relation. A module can be defined as a synchronous or asynchronous composition of other modules. In this report, we use only synchronous composition, meaning that each base module makes a transition simultaneously and in lock step.

3 A Formal Model of a Group-based Quarantine Algorithm

We now use the SAL modeling language to formalize and examine the properties of a group-based worm quarantine defense. This algorithm was first proposed in [BP05], where its behavior was studied in the presence of random-scanning worms through a series of network simulation experiments. The algorithm is itself an integrated combination of two quarantine techniques: connection-base rate limiting (RL), and a group-sharing and corroborative defense protocol we refer to as a leap-ahead (LA) defense, as the broadcasting of alerts among group members allows uninfected segments of a network to anticipate impending infection prior to attack.

Architecturally, the RL and LA components are integrated together into the egress routers of each participating local area network (LAN). RL systems operate under the premise that aggressively propagating worms cause infected hosts to accelerate the rate at which they initiate outbound connections to distinct addresses. The RL component both imposes a limit on the number of

outbound connection targets a local host may attempt to reach per unit time, and informs the LA component when local hosts violate this threshold. Each LA component implements a group sharing protocol to broadcast alerts regarding RL threshold violations and monitors RL violations among its peers. When the LA component observes a sufficient amount of corroboration regarding RL alerts, it transitions the egress router to a defensive posture of traffic filtering based on the common attributes reported within the observed RL alerts.

Figure 1 depicts our algorithm of group defense as a flow diagram. Each LA component maintains an overall aggregate alert level score a for its LAN, which is increased by s (severity) upon observation of an RL threshold violation and each time it receives an alert from its peers. The hold-off counter c suppresses repeated origination of alerts for a period of s time units. We thereby prevent a single participant that experiences a stream of threshold violations over multiple time steps from causing the entire group to enter the defensive posture. When a reaches the alert threshold α, traffic filtering is imposed by the egress router. We define $\alpha = r \cdot s$, where parameter r indicates a corroboration level. Both the alert level and hold-off counter decline steadily. Once the alert level a decays to zero, the defensive posture is abandoned.

Group size for the LA component is defined by parameter G, where each egress router preselects $F = G - 1$ peers to whom it will broadcast RL alerts. For the purpose of optimal group coverage for our modeled population, we arrange the numerical identities of all known egress routers in a ring of ascending order, and assign each router the previous $G - 1$ identities in the ring as its peers.

Figure 2 illustrates the composition of our SAL implementation of this group-based worm quarantine algorithm. Three of these modules, `detector`, `defense`, and `infection`, together compose an egress router, where each egress router serves as an abstraction of a local area network. The full network then contains N instantiations of the egress router logic. The other two modules `all_alerts` and `all_infections` control the emergent network state with respect to alert traffic and infection propagation. We next describe each of these modules in detail. Appendix A lists the complete SAL model of our quarantine algorithm and the nondeterministic worm propagation.

3.1 The Egress Router Representing a LAN

With respect to our model, each egress router serves as an abstraction of a local area network that it guards. An egress router is represented as a synchronous composition of three modules: worm detection, worm defense, and worm propagation logic. In SAL, this composition is written with the following notation, where SAL automatically connects variables of the same name together.

```
egressrouter: MODULE = detector || defense || infection ;
```

The three egress router modules `detector`, `defense`, and `infection` are all zero-delay modules and respond immediately to their inputs. This is a consequence of the use of the primed values of variables in the guards.

The `detector` module represents the RL threshold detection mechanism. In our model, the occurrence of one infected end host implies that the LAN is infected, as we do not explicitly model individual end hosts. The `detector` takes one input `infected`, which denotes whether at least one local end host is infected. Assuming that the scanning worm always scans above the threshold, and it is thus `detectable`, then the output variable `limit` is set to the value of the infection status.

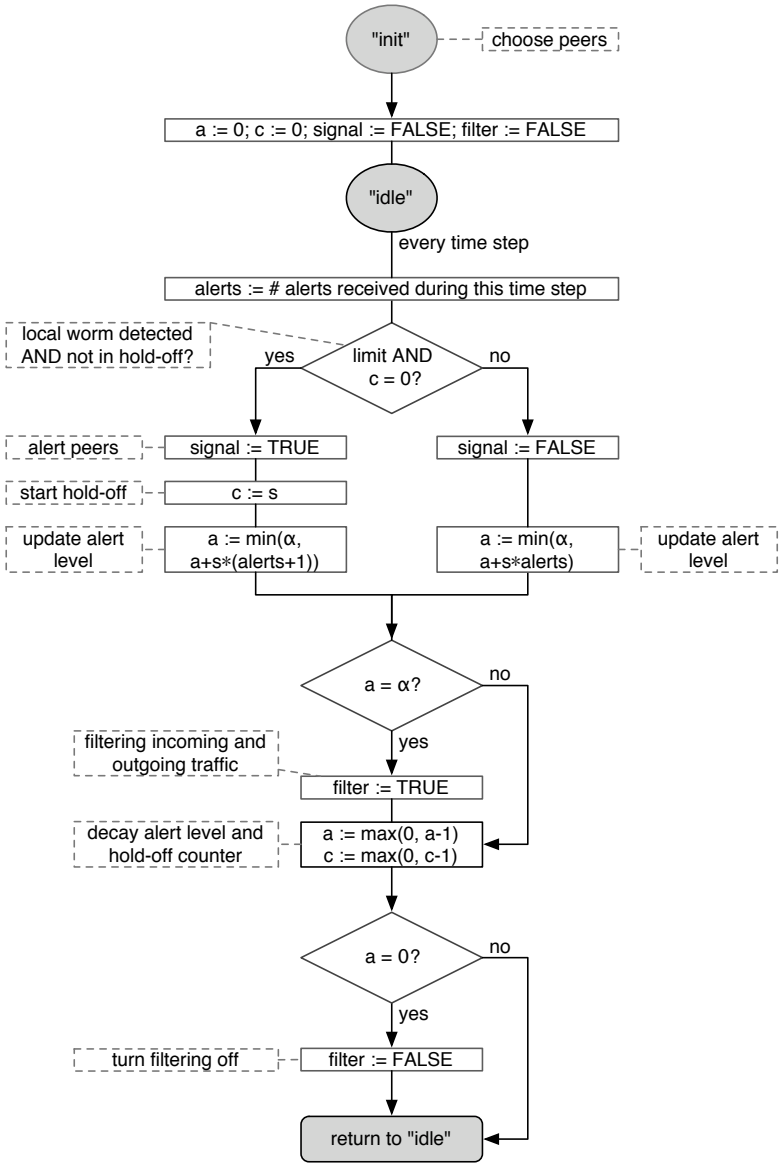

Figure 1: Flow diagram of group-based defense algorithm

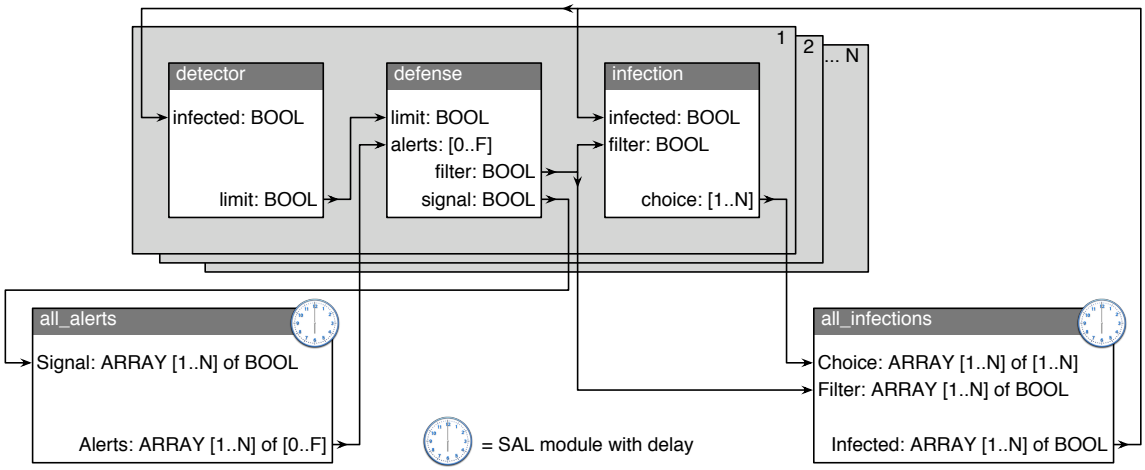

Figure 2: Architecture of formal model in SAL

However, we also model worms that may or may not be detected. Then, the outcome of `limit` is undetermined if there is an infected host present. If none are infected, the output is always `FALSE`.

The `defense` module implements the core of our group-based quarantine algorithm. Two inputs are required. First, `limit` denotes whether the detector has sensed any local worm activity. Second, the module takes the number of `alerts` that fellow peers have generated to warn about possible worm infections. The module produces two outputs. First, it can decide to transition the egress router into or out of the defensive posture, turning the variable `filter` on or off. Second, the defense mechanism may choose to alert its peers about an ongoing infection. In this case, the variable `signal` is set to `TRUE`, whereas `FALSE` implies that the defense has not produced an alert.

Finally, the `infection` module models the worm propagation logic. There are few assumptions regarding the worm's ability to spread. In our model, the defense mechanism is deterministic, but the infection spread mechanism is highly nondeterministic. The `infection` module takes two inputs. First, it needs to know whether the local area network contains an infected host. Second, the infection propagation also depends on whether the `defense` is filtering outbound traffic. In our implementation, if and only if the LAN is infected and the egress router is not filtering, the formal model allows the choice of one of all N possible networks to be attacked next from this one.

3.2 The Full Network

The SAL modules `all_infections` and `all_alerts` represent the interactions of the global network and model the propagation of infection and alerts over the network. Hence, these are *not* zero-delay modules. The module `all_infections` simply collects all the choices infected LANs have made during the epidemic spread. If an egress router of a targeted LAN is not in defensive posture (i.e., is not currently filtering incoming traffic), the LAN is considered susceptible and is therefore infected. Otherwise, the LAN's infection status remains unchanged. The module `all_alerts` accumulates alert

signals generated by all the routers and produces the number of alerts that each egress router will receive in the next time step.

The complete system is described by creating N instances of the `egressrouter` and composing them with one instance each of the `all_alerts` and the `all_infections` modules, as seen in Figure 2. In the full network module, we rename the state variables of the `egressrouter` so that they do not conflict with each other. To do so, we map the state variable, say x, of the i-th router to $X[i]$, where X is an array of the type of x indexed by the router identifiers.

We emphasize that the only nondeterministic choices in the transitions arise from the model of worm propagation. Furthermore, all modules that model computation are zero-delay, whereas modules that model message passing—whether worm infection attempts or defense alerts—over a network induce a one time step delay.

3.3 Model Checking

Once the formal model is defined, we use the SAL model checker to examine the behavioral properties of our algorithm as expressed in the form of linear temporal logic (LTL) [Pnu77]. Apart from the usual boolean operators, LTL also uses two temporal operators, "always" $\Box$ (denoted by `G` in SAL) and "eventually" $\Diamond$ (denoted by `F` in SAL). Once a property of interest is expressed in LTL, the SAL symbolic model checker can analyze the quarantine and worm model state space to determine whether this property holds or is violated, in which case SAL provides the state sequence producing a counter-example.

In our group-based quarantine algorithm, local rate limit threshold violations and alerts received from one's peers are given equivalent weight with respect to the alert level a. In the following sections, we say that if a router received x alerts, then x may include an alert generated by itself, as well as alerts received from others. We also use the terms *router* and *LAN* synonymously with respect to our SAL model. Finally, when attributing router or LAN with the word "infected" or "uninfected," we mean that the associated local area network contains at least one or no infected hosts, respectively.

To apply the model checker, we must instantiate the parameters of our model. N denotes the number of egress routers that represent a local area network behind it, where $N \geq 2$, since by definition one computer is not a network. In this study, we set N to values as high as $N_{max} = 9$. Unfortunately, many SAL model checking runs did not complete for $N_{max} = 9$. All the results stated in the remainder of this report are obtained from completed SAL runs with parameter selections for $2 \leq N \leq N_{max} = 9$.

The variable r determines the amount of corroboration needed before a router will transition its LAN into a defensive posture. The minimum value is $r = 1$, so that a router receiving only one alert switches into defensive posture. The largest value is $r = N$, requiring at least N alerts before a router starts filtering. Third, the group size G denotes the number of selected peers plus one. The value of G ranges between $r \leq G \leq N$ to allow the size of the group to ensure at least an r amount of independent corroboration.

We performed the model checking experiments using the SAL symbolic model checker (`sal-smc`) version 2.4 on two machines with dual 2.8 GHz Xeon processors and 2 GB of memory. Both

computers run the RedHat 7.3 Linux distribution. We aborted those experiments, which had not completed after twelve hours of running.

4 Validating the Model

The intuition behind the group-based quarantine strategy is that as an epidemic spreads, word also spreads from infected to uninfected LANs, which ideally can collect enough peer alerts to corroborate a consistent filtering strategy before they themselves are infected. If an uninfected router receives at least r alerts in the same timestep, the LA component should transition the router into the defensive posture. While in this defensive posture, a previously uninfected router should not become infected as we assume perfect traffic filters for the purposes of our model.

The SAL model checker allows us to postulate this intuition as LTL properties and then either validate or refute them in all possible state transitions between the formal quarantine model and the nondeterministic worm model. For example, we can verify that the quarantine algorithm accurately captures one of our intuitions by checking its formal model against Property 1. Indeed, all experiments that completed within twelve hours proved this property.

Property 1 (Corroboration)**.** *It is always the case that whenever an uninfected router receives at least r alerts at once, then it continues to remain uninfected thereafter. Formally,*

$$\Box(\,\forall j \in \{1..N\} : \mathit{Alerts}[j] \geq r \wedge \neg\, \mathit{Infected}[j] \Rightarrow \Box(\,\neg\, \mathit{Infected}[j]))$$

We further validate our model by determining whether an uninfected LAN's router can ever enter the filtering mode. As stated previously, our group-defense strategy is based on the notion that peer-based sharing of RL alerts will, in the case of scan-based worms that operate above the RL threshold, allow uninfected routers to enter a defensive posture before their LANs are infected. Therefore, we wish to disprove Property 2.

Property 2 (Filtering Implies Infected)**.** *It is always the case that a filtering router is also infected. Formally,*

$$\Box(\,\forall j \in \{1..N\} : \mathit{Filter}[j] \Rightarrow \mathit{Infected}[j])$$

Property 2 holds in only two cases. First, there is the trivial condition where alert propagation is switched off via $G = 1$, which implies that $r = 1$ and that a router "receives" alerts from itself only when it gets infected. At the time of infection, the alert level will reach $\alpha = 1 \cdot s$, pushing the router into an immediate defensive posture. Henceforth, the router remains infected[1] and thus keeps triggering the detector, which in turn prevents the alert level from ever reaching zero again. The model checker also proves Property 2 for the combinations with $r = 2$ and $G = 2$. When $G = 2$, each infected router sends an alert to only one peer, and—in our static group confirmation—each router may receive alerts from only one other peer, and no one else will receive these alerts. Because of this exclusive sharing arrangement and $r = 2$, only infected routers can reach the alert threshold needed to start filtering. In all other experiments, the model checker found counter-examples that disprove Property 2. Figure 3 shows the counter-example constructed by SAL for $N = 3$, $r = 2$, and $G = 3$ that produces a five-step infection and alert sequence, in which router 2 finally starts filtering while not being infected.

[1] We do not model cure.

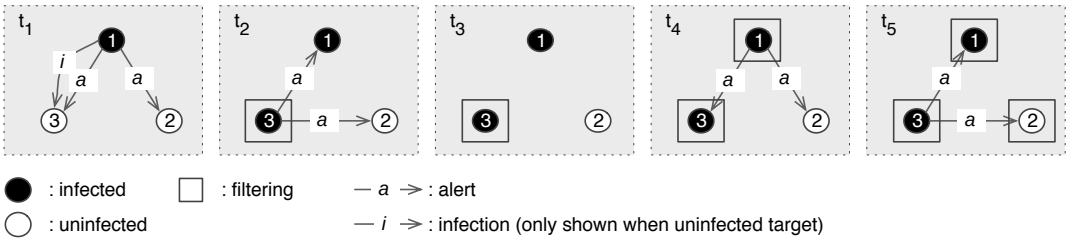

Figure 3: Counter-example for Property 2

Finally, we verify the invariant that once a node starts filtering, it continues to remain in the filtering mode thereafter. Property 3 states this formally.

Property 3 (Remains Filtering)**.** *It is always the case that a filtering router is also filtering henceforth. Formally,*

$$\Box(\forall j \in \{1..N\} : \mathit{Filter}[j] \Rightarrow \Box(\mathit{Filter}[j]))$$

Property 3 holds for all cases except when $r = 1$. Without loss of generality, we set router 1 to be initially infected. With $r = 1$, router 1 switches to filtering mode in the first time step. Then, the alert level a decays simultaneously with all the other parameter adjustments. If the worm is silent and does not infect other routers, which could then alert the initial router, a will reach zero after three time steps and will then switch out of filtering.

5 Formalizing Quarantine

Here, we develop formal definitions of quarantine with respect to our SAL model. We propose that this process of formalizing quarantine can be applied to other models of worm epidemic and infection-filtering quarantine.

Intuitively, successful quarantine means that the infected population is isolated from the rest. In terms of our defense model, we achieve isolation by turning on filtering at egress routers. The LTL formula $\Diamond(\Box(\forall j \in \{1..N\} : \mathrm{Infected}[j] \Rightarrow \mathrm{Filter}[j])$ captures this. We know that infection is an irreversible state,[2] and we have proven[3] that once a router starts filtering it continues to do so. Combining this, we can simplify the formula above to propose Weak Quarantine as follows.

Quarantine 1 (Weak Quarantine)**.** *Eventually every infected router is also filtering. Formally,*

$$\Diamond(\forall j \in \{1..N\} : \mathit{Infected}[j] \Rightarrow \mathit{Filter}[j])$$

Using the SAL model checker, we show that our defense mechanism cannot always reach quarantine. Hence, this property cannot be proven in our model. This is due to the many degrees of freedom that the worm propagation takes on. In our model, the infection spreads nondeterministically, which should not be confused with a random-scanning worm of fixed propagation rate. Our model

[2] We do not consider cure.
[3] For all completed experiments with $r > 1$.

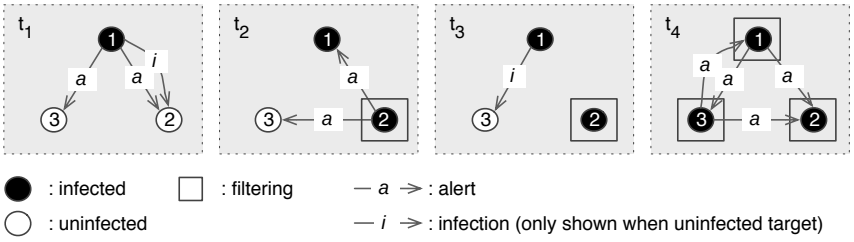

Figure 4: Example when Quarantine 1 holds

encompasses all possible worm strategies with the only constraint of spreading from each infected host network to at most one other network per time step.

Since we cannot prove that our defense always reaches quarantine for every possible worm, we use the model checker to assert the deduction of the negated formula from our `fullnetwork`. We hope to disprove the negation, and counter-examples then guarantee that the original formula holds in these scenarios even in infinite horizon. In contrast, simulations can yield results only about finite horizon.

Figure 4 shows a counter-example to the negation of Quarantine 1 for $N = 3$, $r = 2$, and $G = 3$. In the last time step, Quarantine 1 holds but closer inspection reveals that the proposition does not preclude the whole population from becoming infected. In order to make the definition include the fact that at least one is spared, we propose Beneficial Quarantine as follows.

Quarantine 2 (Beneficial Quarantine)**.** *Eventually every infected router is also filtering and there exists an uninfected router. Formally,*

$$\Diamond(\,(\forall j \in \{1..N\} : \mathit{Infected}[j] \Rightarrow \mathit{Filter}[j]) \wedge (\exists k \in \{1..N\} : \neg\, \mathit{Infected}[k]))$$

We also investigate whether the fact that a router is spared can hold over time. We already discussed that the first part of Quarantine 2 remains to be valid once it becomes true. Therefore, we apply the "always" operator only to the second part.

Quarantine 3 (Beneficial Permanent Quarantine)**.** *Eventually every infected router is also filtering, and henceforth there exists an uninfected router. Formally,*

$$\Diamond(\,(\forall j \in \{1..N\} : \mathit{Infected}[j] \Rightarrow \mathit{Filter}[j]) \wedge \Box(\,\exists k \in \{1..N\} : \neg\, \mathit{Infected}[k]))$$

As expected, the model checker is not able to prove Quarantine 2 or Quarantine 3 for almost all of the given parameter valuations. Only in the cases of $r = 2$ and $G = 2$ are the negations of these propositions proven among the completed experiments. We sketch out the proof for the negation of Quarantine 2, which is stronger than the negation of Quarantine 3, as follows.

Negation of Quarantine 2. It is always the case that there exists an infected but not filtering router or all routers are infected.

Proof. At the beginning, the initially infected router is not filtering and thus satisfies the first part of the negated property. Every infected router sends out one alert to the preceding router at every $s = 3$ time steps. In order to cause any infected router to start filtering, it must receive an alert

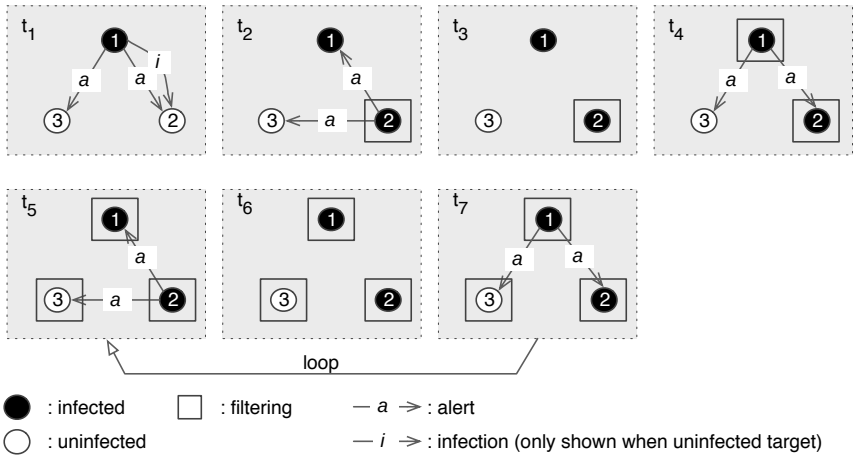

Figure 5: Example when Quarantine 2 (t_1–t_4) and Quarantine 3 (t_1–t_∞) hold

from the next router in line. Furthermore, for $G = 2$, no uninfected router can receive enough alerts to start filtering (we proved this for given parameters in Property 2). Only infected routers send alerts, so when an infected router i switches into filtering mode, we know that its next neighbor $i+1$ has sent an alert and is thus also infected. Now, we apply the same reasoning to this infected neighbor $i+1$: In order to start filtering, it needs to be alerted by its next neighbor $i+2$, which must then be infected as well. Finally, following this reasoning cascade, the ring of infected and filtering routers closes when we arrive at the original router i. Now all routers are infected, which satisfies the second part. ■

Figure 5 shows a typical counter-example of the negation of Quarantine 2 and Quarantine 3. For both properties, the example performs the first four steps until Quarantine 2 holds. Then, the example performs a loop in the following three steps, which guarantees that the spared router never gets infected as required by Quarantine 3.

Next, we strengthen our definition of Beneficial Quarantine to test if the defense mechanism could quarantine the worm without forcing all uninfected nodes into filtering. This expresses the potential of the defense mechanism to be altruistic: the nonfiltering node, despite still allowing all incoming traffic, does not get infected since all infected nodes are employing outbound filtering. Furthermore, similar to Quarantine 3, we can emphasize that once quarantine has been achieved while at least one LAN is spared, this state should continue to hold. We formally state these stronger forms of quarantine as follows.

Quarantine 4 (Strong Quarantine)**.** *Eventually every infected router is also filtering and there exists an uninfected and not filtering router. Formally,*

$$\Diamond(\,(\forall j \in \{1..N\} : \mathit{Infected}[j] \Rightarrow \mathit{Filter}[j]) \wedge (\exists k \in \{1..N\} : \neg\, \mathit{Infected}[k] \wedge \neg\, \mathit{Filter}[k]))$$

Quarantine 5 (Strong Permanent Quarantine)**.** *Eventually every infected router is also filtering, and henceforth there exists an uninfected and not filtering router. Formally,*

$$\Diamond(\,(\forall j \in \{1..N\} : \mathit{Infected}[j] \Rightarrow \mathit{Filter}[j]) \wedge \Box(\,\exists k \in \{1..N\} : \neg\, \mathit{Infected}[k] \wedge \neg\, \mathit{Filter}[k]))$$

As for Quarantine 2 and 3, the negations of Quarantine 4 and 5 hold for cases with $r = 2$ and $G = 2$. The same proof applies here. In addition, the negation of Quarantine 5 is true for $G = N$, which we prove as follows.

Negation of Quarantine 5. It is always the case that there exists an infected but not filtering router or, eventually, all routers are infected or filtering.

Proof. Alerts always come from infected routers. If less than r routers are infected, these cannot be filtering because an infected host needs r alerts to start filtering. This satisfies the first part of negated Quarantine 5. With $G = N$, every alert is received by the whole population. If at least r routers are infected, then, within $s = 3$ time steps, each router—whether infected or not—receives at least r alerts (including its own). After these $a = 3$ time steps, everyone has started filtering, which satisfies the second part. ■

The above formulated types of Strong Quarantine do not hold for arbitrary choice of parameters within the range of our experiments. Again, we searched for counter-examples of the negated formulas. These example executions of the model indicate that in certain scenarios, we can guarantee to achieve Strong Quarantine even in infinite horizon.

Finally, we aim to find a quarantine formula that can be proven in our model. Revisiting the Weak Quarantine formula, we observe that a worm infecting routers that do not exchange alerts can defeat this property. In our defense mechanism, these routers never achieve the corroboration needed to switch into defensive posture. If the worm attempts to propagate further and infects routers that do exchange alerts, then it will trigger the defense mechanism. These observations suggest the following quarantine property, capturing that not being able to achieve Weak Quarantine requires the worm to voluntarily spare at least one LAN.

Quarantine 6 (Quarantine Or Saving One)**.** *Eventually every infected router is also filtering, or henceforth there exists an uninfected router. Formally,*

$$\Diamond(\,(\forall j \in \{1..N\} : \mathit{Infected}[j] \Rightarrow \mathit{Filter}[j]) \vee \Box(\,\exists k \in \{1..N\} : \neg\, \mathit{Infected}[k]))$$

The SAL model checker proves Quarantine 6 in all completed experiments of any parameter valuation. Quarantine 6 indicates that our defense mechanism can help in preventing the worm from achieving its full saturation potential. We establish this result without making any assumption on the way the worm spreads, and hence the claim is true irrespective of the worm's propagation strategy within the limit of replication speed.

6 Designing Smart Worms: Game Theoretic View of Worm Defense

We now investigate the game theoretic aspects of our group-based quarantine algorithm versus a propagating worm. In our model, the moves (transitions) of the defense mechanism are deterministic. The moves of the worm, on the other hand, are nondeterministic. Thus, the worm can possibly defeat the defense mechanism by choosing a smart strategy for infecting the various parts of the network.

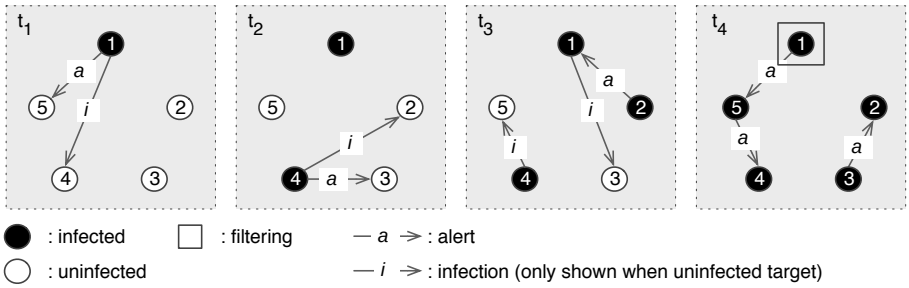

Figure 6: Example of a smart worm with $r = 2$ and $G = 2$

We construct winning strategies for the worm using the SAL model checker. Let us say that the worm wins if it can infect all the routers. Then, counter-examples of the following property show the moves a smart worm should make to defeat our modeled quarantine strategy. However, if model checking proves Property 4, then the worm cannot force the system into a state where all routers are infected.

Property 4 (Defense Wins)**.** *It is always the case that there exists an uninfected router. Formally,*

$$\Box(\exists j \in \{1..N\} : \neg\, Infected[j])$$

For $r = 1$ and $G = 1$, no alerts are generated and it is easy to see that the worm wins as only the routers of infected LANs have a chance to switch into filtering mode. However, for $r = 1$ and $G > 1$, the SAL model checker proves Property 4. In these cases, the initially infected router sends an alert to at least one other router in the first time step, which forces recipients immediately into filtering ($r = 1$) before they are infected. These alerts repeat with periodicity $s = 3$ before routers back off, so that they remain filtering from then on. The worm then never has a chance to infect these routers, and hence no winning strategy for the worm exists.

For $r > 1$, there generally exists a winning strategy for the worm. A few peculiar combinations of $r = 2$ and the other two parameters preclude the worm from winning. Within the experiments that we were able to complete, these combinations are (all with $r = 2$): $G = N = 6$, $G = N = 7$, $G = 7$ and $N = 8$, $G = N = 8$, and $G = N = 9$. We suspect that for $r > 2$, Property 4 can be proven if G and N are sufficiently large. Unfortunately, the performance limitations of the model checker prohibit the full exploration of a state space that large. In future studies, we hope to construct and (manually) prove the general formula for cases in which Property 4 holds.

Figures 6 and 7 exemplify one strategy a "smart worm" can take to counter quarantine.[4] For a small group size (as seen in Figure 6), the worm begins propagating slowly, infecting in the whole network only one new victim per time step and avoiding those that will receive alerts according to the deterministic group selection algorithm. At t_3, when at least half of the population is infected but none of these are filtering, the worm launches a concerted attack against the remaining uninfected population, where every infected router targets a different susceptible router. For a large group size (as seen in Figure 7), the smart worm lies dormant in the first time step. This prevents any new infection from coinciding with the inevitable alerts that the initially infected router sends

[4]Quarantine in the sense that at least one LAN is spared.

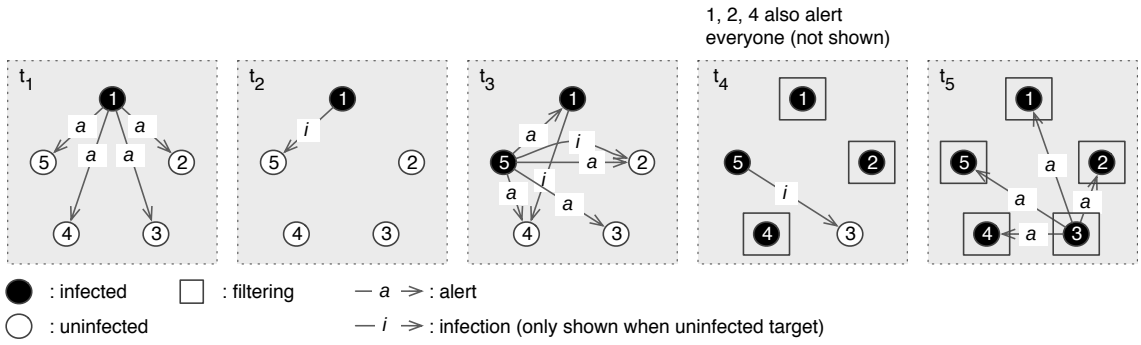

Figure 7: Example of a smart worm with $r = 2$ and $G = 5$

at t_1, so that no one switches into filtering. Then, the worm attack appears again to be concerted. In the next time steps, infected routers choose targets carefully not to be overlapping with others, and reach full saturation before quarantine takes effect.

Essentially, this worm operates in two phases: strategic placement of worms such that infections avoid group overlap, followed by a coordinated overrun of the remaining LANs. While the attack seems to require sophisticated coordination by the worm, this strategy could be precalculated assuming the worm acquires access to the preselected group map. Each worm instantiation is subsequently handed an "infection schedule" of which LANs it is allowed to infect and at what time interval it is allowed to attack. These counter-quarantine scenarios suggest the danger of static preselection of the group structure. One solution is to periodically and randomly alter the set of peers to prevent worms from reliably avoiding group overlap. COVERAGE [AGI+03] is an example of a quarantine strategy that randomly recomputes the group to share alerts at periodic intervals.

In a second strategy, the worm exploits the decay of the alert level by introducing "strategic dormancy" between outside infection attempts. Upon each infection of other LANs, the worm enforces a period of noninfection equal to the value of s. When a dormancy period is completed, the worm can either propagate incrementally across the susceptible network population, or employ knowledge of the quarantine group structure to perform multiple infections of LANs that do not share alerts. Thus, a second quarantine algorithm adjustment may introduce indeterminism in the decay function of c or a, preventing the worm from calculating a safe dormancy period.

Examining the counter-examples produced for many of the parameter combinations indeed exhibits worm behavior that outsmarts our defense scheme and offers an important insight into better algorithm designs. For those algorithms already designed, model checking can inform directed simulation experiments or test case generation, which today does not often include the search for intelligent counter-quarantine strategies.

7 Limitations of Formal Model

The applications of formal modeling and the use of state-transition-based model checking in the design stage of our quarantine algorithm offer several valuable benefits. The SAL environment provides a rapid assessment of our design, the ability to formalize desired properties, and the ability to validate or refute these properties in an efficient and repeatable manner. The model checker also identifies the specific infection sequences and conditions that can effectively counter the quarantine algorithm, which can later be used to inform directed testing or simulation experiments.

However, formal modeling and model checking also has significant limitations, particularly with respect to issues of network abstraction. The semantics of the network infrastructure, such as firewalls, routers, and end hosts, are significantly abstracted, as are the influences of traffic dynamics, congestion, quarantine message volume, or other network characteristics that affect worm propagation speed. Topology complexity and size are also limited, as they dramatically impact memory and computational complexity.

We further bound the infection growth rate to that of exponentially doubling at each time step. In our analysis, the network is represented by a set of egress routers in a fully connected topology. Each egress router serves as an abstraction of a LAN, where the internal infections collectively produce an average outbound spread rate of one infection per time interval. However, in practice worms can spread at faster rates through a number of ways. The average rate of outbound successful infections can increase relative to the number of infected internal hosts that reside behind the router. Also, each infected host could accelerate its outbound scans per time step. However, our outbound-connection rate limiter is specifically designed to throttle high-speed scanning worms. Infection growth rate is also influenced directly by the infection method, depending on the availability of a susceptible and unprotected population.

There are also practical advantages and disadvantages to employing a deterministic group selection mechanism in our quarantine algorithm. On the one hand, a static preselection of peers—whether at random or in a deterministic fashion—requires less coordination among the participants during operation. On the other hand, in Section 6 it becomes apparent that a worm designer who is aware of the preselected group structure could devise an infection sequence that prevents quarantine. The computer security fallacy of *security by obscurity* points out that relying on the secrecy of the preselected group structure is a losing proposition. One solution is to periodically recompute the group structure at random, which prevents one from calculating a counter-quarantine infection sequence. Unfortunately, model-checking a nondeterministic quarantine algorithm against a nondeterministic worm both increases the state space to be searched (possibly into a size not manageable) and makes it impossible to attribute any counter-examples found to the worm behavior.

8 Related Work

Over the last decade, large-scale malicious code epidemics have evolved from rare nuisance applications and research curiosities into a critical information-based security threat. This has inspired an active field of research to study the dynamics of malicious code propagation and strategies to counter epidemic spread. This section summarizes four significant directions in malcious code de-

fense strategies, followed by a summary of the current approaches used to evaluate the effectiveness of these defenses.

8.1 Worm Defense Strategies

Substantial effort has been performed in techniques that we classify as *Resource Limiting (RL) Solutions.* RL solutions explore ways in which local systems and domains may delay worm propagation through the limiting of resources that aggressive worms are known to consume at high rates. Williamson [Wil02] suggests that throttling the volume of outbound connections that a host is allowed to initiate to new machines can produce a significant reduction in the infection rate, without significantly hindering normal communications. Staniford [Staar] refines the outbound connection-throttling concept and provides extensive assessment of its behavior, while moving the throttling mechanism from the individual host to the domain gateway. Gualtieri and Mossé [GM03] propose to dynamically calculate outbound connection rate limits on a per-process basis, through the observation of connection rates across the total population of processes or from a preselected group of known benevolent processes. Ganger et al. [GEB02] suggest that the analysis of network connections not facilitated through DNS lookups provides a relevant signature for identifying potential worm traffic, and host-layer filters can be directed to such traffic with greater aggressiveness. Wong et al. [WWS+04] explore the application of connection rate limiting to backbone routers, suggesting that the throttling of IP-to-IP connection at the edge offers propagation reduction equivalent to all hosts implementing rate throttling, while offering significant deployment advantages.

A second major direction has been toward the design of cooperative information sharing, either hierarchically or using peer-based models, to help recognize the emergence of a propagating worm and then take coordinated action before the worm can saturate the network. We broadly classify these schemes as *Leap Ahead (LA) Solutions,* as they seek to spread warning to network segments not yet affected, and thus potentially prevent the worm from reaching its full saturation potential, assuming a finite time-to-patch interval. For example, Nojiri et al. [NRL03] propose a cooperative alert sharing scheme using a "Friends protocol" under which each node (domain gateway) preselects a set of friends with which to share worm indicators, and in turn is also selected by other domains to receive reports. Alternatively, Anagnostakis et al. [AGI+03] propose a variation of this sharing scheme called COVERAGE, in which a node randomly selects a set of remote nodes to poll for worm reports at periodic intervals.

Predesigned-Preventative (PP) Solutions refer to approaches designed to disrupt or thwart the discovery of susceptible nodes within an address space, potentially by dynamically altering the connectivity of networks or end nodes in the presence of a propagating threat. Briesemeister et. al. [BLP03] study percolation theory or epidemic spread in artificial scale-free networks to suggest how networks could be designed to delay the spread of propagating malicious code while still maintaining high reliability of network links. Gorman et.al. [GKSS03] also examine the use of scale-free properties within the Internet's autonomous system (AS) map, and similarly suggest that the concentration of worm filtering services on the nodes with the highest connection density would yield the greatest return while disrupting the minimum set of network devices. Staniford's work on CounterMalice [Staar] and the study by Zou et al. [ZTG04] of "firewall network systems" explore the preplacement of devices within an enterprise that could facilitate its rapid isolation into subnetworks, thwarting a worm's ability to propagate by preplanning segmentation strategy.

Provos [Pro04] suggests the deployment of honeypot devices in a network that engage in slow connection dialogs as a method to dramatically slow an aggressive worm's ability to discover susceptible hosts within an address space. Wang et al. [WGSZ04] propose the placement of predetermined filters within end node network stacks, which can be rapidly activated as a first line of defense when vulnerabilities are initially discovered and before patches are installed.

Another variation of worm defense involves an active strategy of interception and rapid patching, which Nicol [NL04] refers to as "taking the battle to the worm." We classify these techniques as *Mobile Combat Solutions.* One approach proposes to eliminate propagating malicious mobile code by distributing a mobile self-replicating code module that searches out for signs of a malicious resident code and vaccinates infected machines through a patch or another removal method. For example, Toyoizumi and Kara [TK02] present an analysis of a predatory vaccination application called Predator. The paper employs the biologically inspired "Lotka-Volterra" equation to model the interaction of the predator-prey relationship between the malicious code and mobile predator vaccination, with the goal of minimizing the number of predators required to eliminate the virus threat. The paper suggests that a small number of good predators, on the order of a few thousand could contain an aggressive large-scale worm such as Code Red [MSVS03]. Nicol [NL04] explores four active defense propagation models, from simple scanning systems that race against worms to patch susceptible hosts, to sniper worms that behave similarly to the Predator model. In this study, we examine the complementary nature of two cyber defense strategies: Connection Rate Limiting representing an RL solution, and the Friends protocol, representing an LA solution. Next, we briefly describe the details of these algorithms as applied in our experiment, and discuss the development of a novel combination strategy that overlays both strategies.

8.2 Worm Defense Evaluation

Most of the work in understanding the behavior of malicious code propagation and defense has centered around infection growth rate potential and reduction. Formalizing our understanding of infection rates has revolved around differential equations to describe the epidemic spreading and the impact of the given defense at high levels of abstraction. The use of biologically inspired epidemic-spreading equations in computer virus analysis was presented in [KW91], and since then numerous worm countermeasures studies have incorporated variations of these equations to articulate the benefits of their solutions.

Researchers often turn to simulation as the basis from which to study properties of a proposed malicious code defense [AR05, LYPN02, BP05, ZWZM04]. Once an algorithm is designed, simulation offers a time- and cost-efficient way to examine infection growth rate impacts. Like the formal modeling activity here, simulation incorporates a tremendous amount of abstraction. Simulation typically abstracts end hosts, topology, and worm behavior in ways that do not capture the complex dynamics of the network traffic or infrastructure, though some simulation environments are emerging to help worm simulations better capture traffic dynamics and other operational characteristics [LNBG03].

At the other end of the spectrum, operational testing can provide detailed insight regarding the pragmatic issues of worm defense overhead, management, and impact to normal operations, as well as a greater understanding of the duress that the malicious code outbreak causes. However, testing is generally recognized as being lengthy and expensive, particularly in the assessment of defenses

that are intended to scale and span over large networks. Wide variability of topology configurations and worm behavior is also expensive to fully explore, and testing is often more effective in answering direct questions about specific environments and test cases than in assessing behavioral properties across a range of conditions. Emulation environments may offer a middle ground for worm defense analysis that is closer to reality than formal modeling or simulations. The DETER secure emulation environment [BBB+04] seeks to reduce the cost of creating complex worm emulation experiments that can capture more accurate algorithm implementation and produce real traffic, while reducing the effort and equipment costs associated with live testing.

9 Conclusion

To date, the analysis of worm quarantine algorithms through testing and simulation has provided good insight for understanding their potential impact on the growth rates of certain classes of worm epidemics. However, these techniques do not address the underlying subtleties of what infection quarantine means in the context of network protection, nor do they provide insight into behavioral properties that extend beyond their observation time horizon given completely nondeterministic infection sequences.

This report takes a first step toward formalizing desired properties of a group-based worm quarantine algorithm. Using the SAL formal modeling language and model checker, we define increasingly stronger and permanent forms of quarantine properties, and find cases where our algorithm can and cannot prove these properties. Indeed, the most idealized permanent forms of worm quarantine elude our algorithm. However, given our modeled network and imposing no constraints on the infection sequence, we can prove that the only way a worm can avoid having all of its infected nodes enter filtering mode is for it to stop infecting new nodes.

In addition, we observe that the counter-examples produced by our model checker can provide valuable insight into counter-quarantine strategies that could enable more intelligent worms to circumvent a known quarantine algorithm. These insights could also lead to more directed simulation or operational test generation, although at present that work would most likely be ad hoc and manual.

We motivate this work using a specific previously published algorithm, but can envision generalizing and extending quarantine model checking in the future to include other proposed quarantine designs and worm propagation techniques. We believe the quarantine properties described here are already generalizable to other infection-filtering quarantine algorithms. Finally, applying the model checker to a winning situation of the defense and then generating counter-examples provides infection sequences, which can be interpreted as a trace of a potentially smart worm defeating the modeled defense scheme.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. ANI-0335299, through a subcontract with the University of California at Davis, Contract No. 01RA0052.

References

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

[AR05] M. Abdelhafez and G.F. Riley. Evaluation of worm containment algorithms and their effect on legitimate traffic. In *Third IEEE International Workshop on Information Assurance (IWIA)*, March 2005.

[BBB+04] R. Bajcsy, T. Benzel, M. Bishop, B. Braden, C. Brodley, S. Fahmy, S. Floyd, W. Hardaker, A. Joseph, G. Kesidis, K. Levitt, B. Lindell, P. Liu, D. Miller, R. Mundy, C. Neuman, R. Ostrenga, V. Paxson, P. Porras, C. Rosenberg, J. D. Tygar, S. Sastry, D. Sterne, and S. F. Wu. Cyber defense technology networking and evaluation. *Commun. ACM*, 47(3), 2004.

[BLP03] Linda Briesemeister, Patrick Lincoln, and Phillip Porras. Epidemic profiles and defense of scale-free networks. In *Proceedings of the 2003 ACM Workshop on Rapid Malcode (WORM)*, pages 67–75. ACM Press, October 2003.

[BP05] Linda Briesemeister and Phillip Porras. Microscopic simulation of a group defense strategy. In *Proceedings of Principles of Advanced and Distributed Simulation (PADS)*, June 2005.

[dMOR+04] L. de Moura, S. Owre, H. Rueß, J. Rushby, N. Shankar, M. Sorea, and A. Tiwari. SAL 2. In R. Alur and D. Peled, editors, *Computer-Aided Verification, CAV*, volume 3114 of *LNCS*, pages 496–500. Springer, July 2004.

[GEB02] Gregory R. Ganger, Gregg Economou, and Stanley M. Bielski. Self-securing network interfaces: What, why and how. Technical report, Computer Science Department, Carnegie Mellon University, 2002.

[GKSS03] Sean P. Gorman, Rajendra G. Kulkarni, Laurie A. Schintler, and Roger R. Stough. Least effort strategies for cybersecurity. Technical report, George Mason University, 2003.

[GM03] M. Gualtieri and D. Mossé. Limiting worms via QoS degradation. Technical report, Computer Science Department, University of Pittsburgh, 2003.

[KW91] Jeffrey O. Kephart and Steve R. White. Directed-graph epidemiological models of computer viruses. In *IEEE Symposium on Security and Privacy*, 1991.

[LNBG03] Michael Liljenstam, David M. Nicol, Vincent H. Berk, and Robert S. Gray. Simulating realistic network worm traffic for worm warning system design and testing. In *Proceedings of the 2003 ACM workshop on Rapid Malcode (WORM)*, pages 24–33. ACM Press, 2003.

[LYPN02] Michael Liljenstam, Yougu Yuan, Brian J. Premore, and David M. Nicol. A mixed abstraction level simulation model of large-scale Internet worm infestations. In *10th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 109–116. IEEE Computer Society, 2002.

[MSVS03] D. Moore, C. Shannon, G.M. Voelker, and S. Savage. Internet quarantine: Requirements for containing self-propagating code. In *Proceedings of the 2003 IEEE Infocom Conference (INFOCOM)*, April 2003.

[NL04] D. Nicol and M. Liljenstam. Models of active worm defenses. In *Proceedings of the Measurement, Modeling and Analysis of the Internet Workshop (IMA)*, January 2004.

[NRL03] D. Nojiri, J. Rowe, and K. Levitt. Cooperative response strategies for large scale attack mitigation. In *Proceedings of the 3rd DARPA Information Survivability Conference and Exposition*, April 2003.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, pages 46–67, 1977.

[Pro04] Niels Provos. A virtual honeypot framework. In *Proceedings of the 12th USENIX Security Symposium*, pages 1–14, August 2004.

[SAL03] The SAL intermediate language, 2003. Computer Science Laboratory, SRI International, Menlo Park, CA. `http://sal.csl.sri.com/`.

[Staar] Stuart Staniford. Containment of scanning worms in enterprise networks. *Journal of Computer Security*, to appear.

[TK02] Hiroshi Toyoizumi and Atsuhi Kara. Predators: Good will mobile codes combat against computer viruses. In *Proceedings of the 2002 Workshop on New Security Paradigms (NSPW)*, pages 11–17, 2002.

[WGSZ04] H. Wang, C. Guo, D. Simon, and A. Zugenmaier. Shield: Vulnerability-driven network filters for preventing known vulnerability exploits. Technical report, Microsoft Research, Technical Report MSR-TR-2003-81, 2004.

[Wil02] Matthew M. Williamson. Throttling viruses: Restricting propagation to defeat malicious mobile code. In *Proceedings of the 18th Annual Computer Security Applications Conference*, page 61. IEEE Computer Society, 2002.

[WWS+04] Cynthia Wong, Chenxi Wang, Dawn Song, Stan Bielski, and Gregory R. Ganger. Dynamic quarantine of Internet worms. In *Proceedings of the International Conference on Dependable Systems and Networks DSN-2004*, June 2004.

[ZTG04] Cliff C. Zou, Don Towsley, and Weibo Gong. A firewall network system for worm defense in enterprise networks. Technical Report TR-04-CSE-01, University of Massachusetts Amherst, College of Engineering, February 2004.

[ZWZM04] Yun-Kai Zhang, Fang-Wei Wang, Yu-Qing Zhang, and Jian-Feng Ma. Worm propagation modeling and analysis based on quarantine. In *Proceedings of the 3rd International Conference on Information Security (InfoSecu)*, pages 69–75, 2004.

A Complete SAL Model

```
%% SAL model of worm defense
%% N : number of routers/LANs
%% R : corroboration level
%% Gsize : size of group
%% detectable : if worm can be detected by rate limiter

model{; N : NATURAL, R: NATURAL, Gsize : NATURAL, detectable : BOOLEAN }: CONTEXT =
BEGIN

S : NATURAL = 3 ;             %% severity level of alerts
ALPHA : NATURAL = R*S ;       %% threshold for filtering
Fsize : NATURAL = Gsize - 1 ; %% number of peers

Router_Id : TYPE = [1..N] ;
Router_Set : TYPE = ARRAY Router_Id of BOOLEAN ;

%% helper functions:

next(i: Router_Id, j: NATURAL): Router_Id =
  IF (i + j > N) THEN (i + j - N) ELSE (i + j) ENDIF;

count(i: Router_Id, alerts: Router_Set, fsize: [0..Fsize]): [0..Fsize] =
  IF (fsize = 0) THEN 0
  ELSIF (alerts[next(i,fsize)])
  THEN 1 + count(i, alerts, fsize-1)
  ELSE count(i, alerts, fsize-1)
  ENDIF;

%% THIS MODULE IS ZERO-DELAY. Responds immediately to the inputs.
detector: MODULE =
 BEGIN
  INPUT infected : BOOLEAN    %% any of my nodes are infected?
  OUTPUT limit : BOOLEAN      %% reached rate limit?
  INITIALIZATION
    limit = FALSE;
  TRANSITION
    [
     NOT(infected') -->
      limit' = FALSE          %% no worm -> no signal generated
     []
     infected' AND detectable -->
      limit' = TRUE           %% detectable worm is always detected
     []
     infected' AND NOT(detectable) -->
      limit' IN {TRUE,FALSE}  %% non-detectable worm may be detected
    ]
 END;

%% THIS MODULE IS ZERO-DELAY. Responds immediately to the inputs.
defense: MODULE =
 BEGIN
  LOCAL a : [0..ALPHA]
  LOCAL c : [0..S]
```

```
  INPUT limit : BOOLEAN       %% reached rate limit?
  INPUT alert : [0..Fsize]    %% number of alert messages from peers
  OUTPUT filter : BOOLEAN     %% start filtering?
  OUTPUT signal : BOOLEAN     %% generate alert message for peers?
  INITIALIZATION
      a = 0;
      c = 0;
      signal = FALSE;
      filter = FALSE
  TRANSITION
  [
    limit' AND (c = 0) -->    %% worm detected and not in hold-off
      signal' = TRUE;
      c' = S - 1;
      a' = IF (a+(alert'+1)*S > ALPHA) THEN ALPHA ELSE a+(alert'+1)*S-1 ENDIF;
      filter' = IF (a+(alert'+1)*S >= ALPHA) THEN TRUE ELSE filter ENDIF
    []
    NOT(limit' AND (c = 0)) AND alert' > 0 -->
      signal' = FALSE;
      a' = IF (a+alert'*S > ALPHA) THEN ALPHA ELSE a+alert'*S-1 ENDIF;
      c' = IF (c > 0) THEN c-1 ELSE 0 ENDIF;
      filter' = IF (a+alert'*S >= ALPHA) THEN TRUE ELSE filter ENDIF
    []
    else -->                  %% decay in all other cases
      signal' = FALSE;
      a' = IF (a > 0) THEN a-1 ELSE 0 ENDIF;
      c' = IF (c > 0) THEN c-1 ELSE 0 ENDIF;
      filter' = IF (a = 0 OR a = 1) THEN FALSE ELSE filter ENDIF
  ]
 END;

%% THIS MODULE IS ZERO-DELAY. Responds immediately to the inputs.
infection: MODULE =
 BEGIN
  OUTPUT choice : Router_Id   %% next infection attempt
  INPUT filter : BOOLEAN      %% is this router filtering?
  INPUT infected : BOOLEAN    %% is this router infected?
  INITIALIZATION
      choice = 1
  TRANSITION
  [
    infected' AND NOT(filter') -->
      choice' IN {x : Router_Id | 1 <= x AND x <= N }
  []
    else -->
      choice' = 1             %% choose one known to be infected already
  ]
 END;

%% Infection propagation: NOT ZERO-DELAY.
all_infections : MODULE =
 BEGIN
  INPUT Choice : ARRAY Router_Id of Router_Id   %% choices of nodes
  INPUT Filter : Router_Set                    %% which nodes are filtering
  OUTPUT Infected : Router_Set                 %% infected nodes
  INITIALIZATION
      Infected = [ [j: Router_Id] FALSE] WITH [1] := TRUE
  TRANSITION
```

```
      Infected' = [ [j: Router_Id]
            IF (NOT(Filter[j]) AND (exists(k: Router_Id): Choice[k] = j))
            THEN TRUE
            ELSE Infected[j]
            ENDIF ]
 END;

%% Alert propagation: NOT ZERO-DELAY.
all_alerts : MODULE =
 BEGIN
  INPUT Signal : Router_Set                    %% source alert signals
  OUTPUT Alerts : ARRAY Router_Id of [0..Fsize] %% propagated alerts
  INITIALIZATION
      Alerts = [ [j: Router_Id] 0 ]
  TRANSITION
      Alerts' = [ [j: Router_Id] count(j, Signal, Fsize) ]
 END;

egressrouter: MODULE = detector || defense || infection ;

fullnetwork : MODULE =
 ((WITH INPUT Infected : Router_Set;
      OUTPUT Filter : Router_Set;
      OUTPUT Signal : Router_Set;
      INPUT Alerts : ARRAY Router_Id of [0..Fsize];
      OUTPUT Limit: Router_Set;
      OUTPUT Choice: ARRAY Router_Id of Router_Id
      ( || (j: Router_Id) : RENAME infected to Infected[j],
                  filter to Filter[j],
                  signal to Signal[j],
                  alert to Alerts[j],
                  limit to Limit[j],
                  choice to Choice[j]
            IN egressrouter))
 ||
 all_infections || all_alerts) ;

%% Properties:

corroboration: THEOREM
  fullnetwork |- G( forall(j: Router_Id):
                  Alerts[j] >= R AND NOT(Infected[j])
                  =>
                  G( NOT(Infected[j]) ) );

filtering_implies_infected: THEOREM
  fullnetwork |- G( (forall(j: Router_Id): Filter[j] => Infected[j] ));

remains_filtering: THEOREM
  fullnetwork |- G( (forall(j: Router_Id): Filter[j] => G( Filter[j] )) );

N_weak_quarantine: THEOREM
  fullnetwork |- G( (exists(j: Router_Id): Infected[j] AND NOT(Filter[j])) );

N_beneficial_quarantine: THEOREM
  fullnetwork |- G( (exists(j: Router_Id): Infected[j] AND NOT(Filter[j])) OR
                    (forall(k: Router_Id): Infected[k]) );
```

```
N_beneficial_permanent_quarantine: THEOREM
  fullnetwork |- G( (exists(j: Router_Id): Infected[j] AND NOT(Filter[j])) OR
                   F( forall(k: Router_Id): Infected[k] ) );

N_strong_quarantine: THEOREM
  fullnetwork |- G( (exists(j: Router_Id): Infected[j] AND NOT(Filter[j])) OR
                   (forall(k: Router_Id): Infected[k] OR Filter[k]) );

N_strong_permanent_quarantine: THEOREM
  fullnetwork |- G( (exists(j: Router_Id): Infected[j] AND NOT(Filter[j])) OR
                   F( forall(k: Router_Id): Infected[k] OR Filter[k] ) );

quarantine_or_saving1: THEOREM
  fullnetwork |- F( (forall(j: Router_Id): Infected[j] => Filter[j]) OR
                   G( exists(j: Router_Id): NOT(Infected[j]) ) );

defense_wins: THEOREM
  fullnetwork |- G( (exists(j: Router_Id): NOT(Infected[j])) );

END
```

B Model-Checking Results

The following tables contain the results we obtained from model checking the SAL model shown in Appendix A. To encode the result of a model-checking run, we use this notation.

p : proved

x : counter-example

\- : out-of-memory

b : failed to build counter-example

(gray) : parameter combination not applicable

? : run did not finish

B.1 Property 1 "Corroboration"

G \ N	2	3	4	5	6	7	8	9
1	p	p	p	p	p	p	p	?
2	p	p	p	p	p	p	p	?
3		p	p	p	p	p	p	?
4			p	p	p	p	p	?
5				p	p	p	p	?
6					p	p	p	?
7						p	p	?
8							p	?
9								?

Table 1: Results for Property 1 "Corroboration", $r = 1$

G \ N	2	3	4	5	6	7	8	9
2	p	p	p	p	p	p	p	?
3		p	p	p	p	p	p	?
4			p	p	p	p	-	?
5				p	p	p	p	?
6					p	p	p	?
7						p	p	?
8							p	?
9								?

Table 2: Results for Property 1 "Corroboration", $r = 2$

G \ N	2	3	4	5	6	7	8	9
3		p	p	p	p	-	-	?
4			p	p	p	?	?	?
5				p	p	p	p	?
6					p	p	p	?
7						p	p	?
8							p	?
9								?

Table 3: Results for Property 1 "Corroboration", $r = 3$

G \ N	2	3	4	5	6	7	8	9
4			p	p	p	p	-	?
5				p	p	p	?	?
6					p	p	?	?
7						p	?	?
8							?	?
9								?

Table 4: Results for Property 1 "Corroboration", $r = 4$

G \ N	2	3	4	5	6	7	8	9
5				p	p	p	?	?
6					p	p	?	?
7						p	?	?
8							?	?
9								?

Table 5: Results for Property 1 "Corroboration", $r = 5$

G \ N	2	3	4	5	6	7	8	9
6					p	p	?	?
7						p	?	?
8							?	?
9								?

Table 6: Results for Property 1 "Corroboration", $r = 6$

G \ N	2	3	4	5	6	7	8	9
7						p	?	?
8							?	?
9								?

Table 7: Results for Property 1 "Corroboration", $r = 7$

G \ N	2	3	4	5	6	7	8	9
8							?	?
9								?

Table 8: Results for Property 1 "Corroboration", $r = 8$

G \ N	2	3	4	5	6	7	8	9
9								?

Table 9: Results for Property 1 "Corroboration", $r = 9$

B.2 Property 2 "Filtering Implies Infected"

G \ N	2	3	4	5	6	7	8	9
1	p	p	p	p	p	p	p	p
2	x	x	x	x	x	x	x	x
3		x	x	x	x	x	x	x
4			x	x	x	x	x	x
5				x	x	x	x	x
6					x	x	x	x
7						x	x	x
8							x	x
9								x

Table 10: Results for Property 2 "Filtering Implies Infected", $r = 1$

G \ N	2	3	4	5	6	7	8	9
2	p	p	p	p	p	p	p	-
3		x	x	x	x	x	x	x
4			x	x	x	x	x	x
5				x	x	x	x	x
6					x	x	x	x
7						x	x	x
8							x	x
9								x

Table 11: Results for Property 2 "Filtering Implies Infected", $r = 2$

G \ N	2	3	4	5	6	7	8	9
3		x	x	x	x	x	x	?
4			x	x	x	x	x	?
5				x	x	x	x	?
6					x	x	x	?
7						x	x	?
8							x	?
9								?

Table 12: Results for Property 2 "Filtering Implies Infected", $r = 3$

G \ N	2	3	4	5	6	7	8	9
4			x	x	x	x	?	?
5				x	x	x	?	?
6					x	x	?	?
7						x	?	?
8							?	?
9								?

Table 13: Results for Property 2 "Filtering Implies Infected", $r = 4$

G \ N	2	3	4	5	6	7	8	9
5				x	x	?	?	?
6					x	-	?	?
7						x	?	?
8							?	?
9								?

Table 14: Results for Property 2 "Filtering Implies Infected", $r = 5$

G \ N	2	3	4	5	6	7	8	9
6					x	x	?	?
7						x	x	?
8							x	?
9								?

Table 15: Results for Property 2 "Filtering Implies Infected", $r = 6$

G \ N	2	3	4	5	6	7	8	9
7						x	x	?
8							x	?
9								?

Table 16: Results for Property 2 "Filtering Implies Infected", $r = 7$

G \ N	2	3	4	5	6	7	8	9
8							x	?
9								?

Table 17: Results for Property 2 "Filtering Implies Infected", $r = 8$

G \ N	2	3	4	5	6	7	8	9
9								?

Table 18: Results for Property 2 "Filtering Implies Infected", $r = 9$

B.3 Property 3 "Remains Filtering"

G \ N	2	3	4	5	6	7	8	9
1	x	x	x	x	x	x	x	x
2	x	x	x	x	x	x	x	x
3		x	x	x	x	x	x	x
4			x	x	x	x	x	x
5				x	x	x	x	x
6					x	x	x	x
7						x	x	x
8							x	x
9								x

Table 19: Results for Property 3 "Remains Filtering", $r = 1$

G \ N	2	3	4	5	6	7	8	9
2	p	p	p	p	p	p	p	-
3		p	p	p	p	p	-	-
4			p	p	p	p	p	?
5				p	p	p	p	?
6					p	p	p	-
7						p	p	p
8							p	p
9								p

Table 20: Results for Property 3 "Remains Filtering", $r = 2$

G \ N	2	3	4	5	6	7	8	9
3		p	p	p	p	p	-	?
4			p	p	p	p	-	?
5				p	p	p	p	?
6					p	p	p	?
7						p	p	?
8							p	?
9								?

Table 21: Results for Property 3 "Remains Filtering", $r = 3$

G \ N	2	3	4	5	6	7	8	9
4			p	p	p	-	?	?
5				p	p	p	?	?
6					p	p	?	?
7						p	?	?
8							?	?
9								?

Table 22: Results for Property 3 "Remains Filtering", $r = 4$

G \ N	2	3	4	5	6	7	8	9
5				p	p	-	?	?
6					p	?	?	?
7						p	?	?
8							?	?
9								?

Table 23: Results for Property 3 "Remains Filtering", $r = 5$

G \ N	2	3	4	5	6	7	8	9
6					p	p	?	?
7						p	-	?
8							p	?
9								?

Table 24: Results for Property 3 "Remains Filtering", $r = 6$

G \ N	2	3	4	5	6	7	8	9
7						p	-	?
8							?	?
9								?

Table 25: Results for Property 3 "Remains Filtering", $r = 7$

G \ N	2	3	4	5	6	7	8	9
8							?	?
9								?

Table 26: Results for Property 3 "Remains Filtering", $r = 8$

G \ N	2	3	4	5	6	7	8	9
9								?

Table 27: Results for Property 3 "Remains Filtering", $r = 9$

B.4 Quarantine 1 "Weak Quarantine"

G \ N	2	3	4	5	6	7	8	9
1	x	x	x	x	x	x	?	?
2	x	x	x	x	x	x	?	?
3		x	x	x	x	x	?	?
4			x	x	x	x	?	?
5				x	x	x	?	?
6					x	x	?	?
7						x	?	?
8							?	?
9								?

Table 28: Results for Quarantine 1 "Weak Quarantine", $r = 1$

G \ N	2	3	4	5	6	7	8	9
2	x	x	x	x	x	x	?	?
3		x	x	x	x	x	?	?
4			x	x	x	x	?	?
5				x	x	x	?	?
6					x	x	?	?
7						x	?	?
8							?	?
9								?

Table 29: Results for Quarantine 1 "Weak Quarantine", $r = 2$

G \ N	2	3	4	5	6	7	8	9
3		x	x	x	x	x	?	?
4			x	x	x	x	?	?
5				x	x	x	?	?
6					x	x	?	?
7						x	?	?
8							?	?
9								?

Table 30: Results for Quarantine 1 "Weak Quarantine", $r = 3$

G \ N	2	3	4	5	6	7	8	9
4			x	x	x	x	?	?
5				x	x	x	?	?
6					x	x	?	?
7						x	?	?
8							?	?
9								?

Table 31: Results for Quarantine 1 "Weak Quarantine", $r = 4$

G \ N	2	3	4	5	6	7	8	9
5				x	x	?	?	?
6					x	-	?	?
7						x	?	?
8							?	?
9								?

Table 32: Results for Quarantine 1 "Weak Quarantine", $r = 5$

G \ N	2	3	4	5	6	7	8	9
6					x	x	?	?
7						x	?	?
8							?	?
9								?

Table 33: Results for Quarantine 1 "Weak Quarantine", $r = 6$

G \ N	2	3	4	5	6	7	8	9
7						x	?	?
8							?	?
9								?

Table 34: Results for Quarantine 1 "Weak Quarantine", $r = 7$

G \ N	2	3	4	5	6	7	8	9
8							?	?
9								?

Table 35: Results for Quarantine 1 "Weak Quarantine", $r = 8$

G \ N	2	3	4	5	6	7	8	9
9								?

Table 36: Results for Quarantine 1 "Weak Quarantine", $r = 9$

B.5 Quarantine 2 "Beneficial Quarantine"

G \ N	2	3	4	5	6	7	8	9
1	x	x	x	x	x	x	x	x
2	x	x	x	x	x	x	x	x
3		x	x	x	x	x	x	x
4			x	x	x	x	x	x
5				x	x	x	x	x
6					x	x	x	x
7						x	x	x
8							x	x
9								x

Table 37: Results for Quarantine 2 "Beneficial Quarantine", $r = 1$

G \ N	2	3	4	5	6	7	8	9
2	p	p	p	p	p	p	p	?
3		x	x	x	x	x	x	?
4			x	x	x	x	x	?
5				x	x	x	x	?
6					x	x	x	?
7						x	x	?
8							x	?
9								?

Table 38: Results for Quarantine 2 "Beneficial Quarantine", $r = 2$

G \ N	2	3	4	5	6	7	8	9
3		x	x	x	x	x	x	?
4			x	x	x	x	x	?
5				x	x	x	x	?
6					x	x	x	?
7						x	x	?
8							x	?
9								?

Table 39: Results for Quarantine 2 "Beneficial Quarantine", $r = 3$

G \ N	2	3	4	5	6	7	8	9
4			x	x	x	x	?	?
5				x	x	x	?	?
6					x	x	?	?
7						x	?	?
8							?	?
9								?

Table 40: Results for Quarantine 2 "Beneficial Quarantine", $r = 4$

G \ N	2	3	4	5	6	7	8	9
5				x	x	?	?	?
6					x	?	?	?
7						?	?	?
8							?	?
9								?

Table 41: Results for Quarantine 2 "Beneficial Quarantine", $r = 5$

G \ N	2	3	4	5	6	7	8	9
6					x	?	?	?
7						?	?	?
8							?	?
9								?

Table 42: Results for Quarantine 2 "Beneficial Quarantine", $r = 6$

G \ N	2	3	4	5	6	7	8	9
7						?	?	?
8							?	?
9								?

Table 43: Results for Quarantine 2 "Beneficial Quarantine", $r = 7$

G \ N	2	3	4	5	6	7	8	9
8							?	?
9								?

Table 44: Results for Quarantine 2 "Beneficial Quarantine", $r = 8$

G \ N	2	3	4	5	6	7	8	9
9								?

Table 45: Results for Quarantine 2 "Beneficial Quarantine", $r = 9$

B.6 Quarantine 3 "Beneficial Permanent Quarantine"

G \ N	2	3	4	5	6	7	8	9
1	x	x	x	x	x	x	x	?
2	x	x	x	x	x	x	x	?
3		x	x	x	x	x	x	?
4			x	x	x	x	x	?
5				x	x	x	x	?
6					x	x	x	?
7						x	x	?
8							x	?
9								?

Table 46: Results for Quarantine 3 "Beneficial Permanent Quarantine", $r = 1$

G \ N	2	3	4	5	6	7	8	9
2	p	p	p	p	p	p	p	?
3		x	x	x	x	-	?	?
4			x	x	x	-	-	?
5				x	x	x	-	?
6					x	x	x	?
7						x	?	?
8							?	?
9								?

Table 47: Results for Quarantine 3 "Beneficial Permanent Quarantine", $r = 2$

G \ N	2	3	4	5	6	7	8	9
3		x	x	x	x	?	?	?
4			x	x	x	?	?	?
5				x	x	?	?	?
6					x	?	?	?
7						?	?	?
8							?	?
9								?

Table 48: Results for Quarantine 3 "Beneficial Permanent Quarantine", $r = 3$

G \ N	2	3	4	5	6	7	8	9
4			x	x	-	?	?	?
5				x	-	?	?	?
6					x	?	?	?
7						?	?	?
8							?	?
9								?

Table 49: Results for Quarantine 3 "Beneficial Permanent Quarantine", $r = 4$

G \ N	2	3	4	5	6	7	8	9
5				x	-	?	?	?
6					x	?	?	?
7						?	?	?
8							?	?
9								?

Table 50: Results for Quarantine 3 “Beneficial Permanent Quarantine”, $r = 5$

G \ N	2	3	4	5	6	7	8	9
6					x	?	?	?
7						?	?	?
8							?	?
9								?

Table 51: Results for Quarantine 3 “Beneficial Permanent Quarantine”, $r = 6$

G \ N	2	3	4	5	6	7	8	9
7						?	?	?
8							?	?
9								?

Table 52: Results for Quarantine 3 “Beneficial Permanent Quarantine”, $r = 7$

G \ N	2	3	4	5	6	7	8	9
8							?	?
9								?

Table 53: Results for Quarantine 3 “Beneficial Permanent Quarantine”, $r = 8$

G \ N	2	3	4	5	6	7	8	9
9								?

Table 54: Results for Quarantine 3 “Beneficial Permanent Quarantine”, $r = 9$

B.7 Quarantine 4 "Strong Quarantine"

G \ N	2	3	4	5	6	7	8	9
1	x	x	x	x	x	x	x	x
2	x	x	x	x	x	x	x	x
3		x	x	x	x	x	x	x
4			x	x	x	x	x	x
5				x	x	x	x	x
6					x	x	x	x
7						x	x	x
8							x	x
9								x

Table 55: Results for Quarantine 4 "Strong Quarantine", $r = 1$

G \ N	2	3	4	5	6	7	8	9
2	p	p	p	p	p	p	p	?
3		x	x	x	x	x	x	?
4			x	x	x	x	x	?
5				x	x	x	x	?
6					x	x	x	?
7						x	x	?
8							x	?
9								?

Table 56: Results for Quarantine 4 "Strong Quarantine", $r = 2$

G \ N	2	3	4	5	6	7	8	9
3		x	x	x	x	x	x	?
4			x	x	x	x	x	?
5				x	x	x	x	?
6					x	x	x	?
7						x	x	?
8							x	?
9								?

Table 57: Results for Quarantine 4 "Strong Quarantine", $r = 3$

G \ N	2	3	4	5	6	7	8	9
4			x	x	x	?	?	?
5				x	x	?	?	?
6					x	?	?	?
7						?	?	?
8							?	?
9								?

Table 58: Results for Quarantine 4 "Strong Quarantine", $r = 4$

G \ N	2	3	4	5	6	7	8	9
5				x	x	?	?	?
6					x	?	?	?
7						?	?	?
8							?	?
9								?

Table 59: Results for Quarantine 4 “Strong Quarantine”, $r = 5$

G \ N	2	3	4	5	6	7	8	9
6					x	?	?	?
7						?	?	?
8							?	?
9								?

Table 60: Results for Quarantine 4 “Strong Quarantine”, $r = 6$

G \ N	2	3	4	5	6	7	8	9
7						?	?	?
8							?	?
9								?

Table 61: Results for Quarantine 4 “Strong Quarantine”, $r = 7$

G \ N	2	3	4	5	6	7	8	9
8							?	?
9								?

Table 62: Results for Quarantine 4 “Strong Quarantine”, $r = 8$

G \ N	2	3	4	5	6	7	8	9
9								?

Table 63: Results for Quarantine 4 “Strong Quarantine”, $r = 9$

B.8 Quarantine 5 "Strong Permanent Quarantine"

G \ N	2	3	4	5	6	7	8	9
1	x	x	x	x	x	x	x	x
2	p	x	x	x	x	x	x	x
3		p	x	x	x	x	x	x
4			p	x	x	x	x	x
5				p	x	x	x	x
6					p	x	x	x
7						p	x	x
8							p	x
9								p

Table 64: Results for Quarantine 5 "Strong Permanent Quarantine", $r = 1$

G \ N	2	3	4	5	6	7	8	9
2	p	p	p	p	p	p	p	?
3		p	x	x	x	x	?	?
4			p	x	x	x	x	?
5				p	x	x	x	?
6					p	x	x	?
7						p	x	?
8							p	?
9								?

Table 65: Results for Quarantine 5 "Strong Permanent Quarantine", $r = 2$

G \ N	2	3	4	5	6	7	8	9
3		p	x	x	x	x	?	?
4			p	x	x	x	-	?
5				p	x	-	-	?
6					p	x	-	?
7						p	x	?
8							p	?
9								?

Table 66: Results for Quarantine 5 "Strong Permanent Quarantine", $r = 3$

G \ N	2	3	4	5	6	7	8	9
4			p	x	x	?	?	?
5				p	x	?	?	?
6					p	?	?	?
7						?	?	?
8							?	?
9								?

Table 67: Results for Quarantine 5 "Strong Permanent Quarantine", $r = 4$

G \ N	2	3	4	5	6	7	8	9
5				p	x	?	?	?
6					p	?	?	?
7						?	?	?
8							?	?
9								?

Table 68: Results for Quarantine 5 "Strong Permanent Quarantine", $r = 5$

G \ N	2	3	4	5	6	7	8	9
6					p	?	?	?
7						?	?	?
8							?	?
9								?

Table 69: Results for Quarantine 5 "Strong Permanent Quarantine", $r = 6$

G \ N	2	3	4	5	6	7	8	9
7						?	?	?
8							?	?
9								?

Table 70: Results for Quarantine 5 "Strong Permanent Quarantine", $r = 7$

G \ N	2	3	4	5	6	7	8	9
8							?	?
9								?

Table 71: Results for Quarantine 5 "Strong Permanent Quarantine", $r = 8$

G \ N	2	3	4	5	6	7	8	9
9								?

Table 72: Results for Quarantine 5 "Strong Permanent Quarantine", $r = 9$

B.9 Quarantine 6 "Quarantine Or Saving One"

G \ N	2	3	4	5	6	7	8	9
1	p	p	p	p	p	p	p	?
2	p	p	p	p	p	p	p	?
3		p	p	p	p	p	p	?
4			p	p	p	p	p	?
5				p	p	p	p	?
6					p	p	p	?
7						p	p	?
8							p	?
9								?

Table 73: Results for Quarantine 6 "Quarantine Or Saving One", $r = 1$

G \ N	2	3	4	5	6	7	8	9
2	p	p	p	p	p	p	?	?
3		p	p	p	p	p	-	?
4			p	p	p	p	p	?
5				p	p	p	p	?
6					p	p	p	?
7						p	p	?
8							p	?
9								?

Table 74: Results for Quarantine 6 "Quarantine Or Saving One", $r = 2$

G \ N	2	3	4	5	6	7	8	9
3		p	p	p	p	?	?	?
4			p	p	p	p	?	?
5				p	p	p	?	?
6					p	p	?	?
7						p	?	?
8							?	?
9								?

Table 75: Results for Quarantine 6 "Quarantine Or Saving One", $r = 3$

G \ N	2	3	4	5	6	7	8	9
4			p	p	p	-	?	?
5				p	p	?	?	?
6					p	p	?	?
7						p	?	?
8							?	?
9								?

Table 76: Results for Quarantine 6 "Quarantine Or Saving One", $r = 4$

G \ N	2	3	4	5	6	7	8	9
5				p	p	?	?	?
6					p	?	?	?
7						p	?	?
8							?	?
9								?

Table 77: Results for Quarantine 6 "Quarantine Or Saving One", $r = 5$

G \ N	2	3	4	5	6	7	8	9
6					p	p	?	?
7						p	?	?
8							?	?
9								?

Table 78: Results for Quarantine 6 "Quarantine Or Saving One", $r = 6$

G \ N	2	3	4	5	6	7	8	9
7						p	?	?
8							?	?
9								?

Table 79: Results for Quarantine 6 "Quarantine Or Saving One", $r = 7$

G \ N	2	3	4	5	6	7	8	9
8							?	?
9								?

Table 80: Results for Quarantine 6 "Quarantine Or Saving One", $r = 8$

G \ N	2	3	4	5	6	7	8	9
9								?

Table 81: Results for Quarantine 6 "Quarantine Or Saving One", $r = 9$

B.10 Property 4 "Defense Wins"

G \ N	2	3	4	5	6	7	8	9
1	x	x	x	x	x	x	x	x
2	p	p	p	p	p	p	p	p
3		p	p	p	p	p	p	p
4			p	p	p	p	p	p
5				p	p	p	p	p
6					p	p	p	p
7						p	p	p
8							p	p
9								p

Table 82: Results for Property 4 "Defense Wins", $r = 1$

G \ N	2	3	4	5	6	7	8	9
2	x	x	x	x	x	x	x	x
3		x	x	x	x	x	x	x
4			x	x	x	x	x	x
5				x	x	x	x	x
6					p	x	x	x
7						p	p	x
8							p	p
9								p

Table 83: Results for Property 4 "Defense Wins", $r = 2$

G \ N	2	3	4	5	6	7	8	9
3		x	x	x	x	x	x	?
4			x	x	x	x	x	?
5				x	x	x	x	?
6					x	x	x	?
7						x	x	?
8							x	?
9								?

Table 84: Results for Property 4 "Defense Wins", $r = 3$

G \ N	2	3	4	5	6	7	8	9
4			x	x	x	x	?	?
5				x	x	x	?	?
6					x	x	?	?
7						x	?	?
8							?	?
9								?

Table 85: Results for Property 4 "Defense Wins", $r = 4$

G \ N	2	3	4	5	6	7	8	9
5				x	x	x	?	?
6					x	-	?	?
7						x	?	?
8							?	?
9								?

Table 86: Results for Property 4 “Defense Wins”, $r = 5$

G \ N	2	3	4	5	6	7	8	9
6					x	x	x	?
7						x	x	?
8							x	?
9								?

Table 87: Results for Property 4 “Defense Wins”, $r = 6$

G \ N	2	3	4	5	6	7	8	9
7						x	x	?
8							x	?
9								?

Table 88: Results for Property 4 “Defense Wins”, $r = 7$

G \ N	2	3	4	5	6	7	8	9
8							x	?
9								?

Table 89: Results for Property 4 “Defense Wins”, $r = 8$

G \ N	2	3	4	5	6	7	8	9
9								?

Table 90: Results for Property 4 “Defense Wins”, $r = 9$