Using Predators to Combat Worms and Viruses: A Simulation-Based Study

Ajay Gupta, Daniel C. DuVarney
Department of Computer Science
Stony Brook University
Stony Brook, NY 11794
{ajay,dand}@cs.sunysb.edu

Abstract

Large-scale attacks generated by fast-spreading or stealthy malicious mobile code, such as flash worms and e-mail viruses, demand new approaches to patch management and disinfection. Currently popular centralized approaches suffer from distribution bottlenecks which cannot be solved by merely increasing the number of servers, as the number of servers required to eliminate all bottlenecks is impractically large. Recently, predators were proposed as a technique for eliminating automated mobile malware from computer networks. Predators are benevolent, self-propagating mobile programs which have the ability to clean up systems infected by malignant worms/viruses. We propose a number of extensions to the original predator model, including immunizing predators, persistent predators, and seeking predators. We report on a set of simulations which explore the effects of predators on small-scale (800 to 1600 node) networks. Our results indicate that predators hold significant promise as an alternative to the centralized patch distribution mechanism. The results show that predators can be used to disinfect systems and distribute patches rapidly across the network, without suffering from bottlenecks or causing network congestion. The results also show that the new predator models provide significant benefits over the original predator model. The simulation tool is also useful for tuning predator behavior, so that an optimal tradeoff between the peak virus/worm infection rate and the overhead generated by the predator can be chosen before a predator is released.

1. Introduction

Improving the defensive capabilities of computer networks from self-propagating attacks, such as worms and e-mail viruses, is an urgent problem [4, 7, 6, 10, 8]. The rate at which these types of attacks spread across networks has been steadily increasing, to the point where some recent worms, such as the Blaster worm [9], SoBig [12], Sircam [5] and Code Red [11, 26, 32], have infected hundreds of thousands of computers within a matter of a few hours. The phenomenal propagation rates of these new worms make a rapid response to self-propagating attacks critical.

The reaction to a self-propagating attack can be viewed as a two phase process. In the first phase, the attack is detected, the vulnerability exploited by the attack is diagnosed, and a patch is developed. If the vulnerability is novel (i.e., it was previously unknown), then a temporary patch which disables the service targeted may be employed in order to provide a timely response. Approaches such as automatic patch generation [25] have the potential for quickly generating patches, and other approaches, such as address space randomization [2, 14], can slow down the propagation rate of attacks that exploit memory errors and cause failed attacks to become conspicuous, making early detection more likely and buying time for a patch to be developed. These developments lend credence to the hope that, in many cases, patches will be available before a worm has penetrated much of the network.

In the second phase, malware installed by the worm must be removed and the patch must be applied to all vulnerable machines on the network. In this paper, we address the issues involved in this latter phase.

Current approaches to patch distribution are primarily centralized, and hence suffer from bottlenecks at the server end. Both the server/push approach, in which servers broadcast patches to client machines, and the client/pull approach, in which clients download the patch from a server, suffer from bottlenecks due to centralization. Consider a typical patch of the size of 10Mb to be installed on 10 million machines in an hour, which would require a net bandwidth of 200 Gb/s. While these bottlenecks could potentially be relieved by caching patches across the network, such an approach is a complex solution which requires many additional machines, and furthermore, only reduces the distribution time by a linear amount of time, i.e., the patch distribution time is essentially $O(N/k)$, where N is the number of clients and k is the number of server/cache

machines. Additionally, any centralized approach is vulnerable to a denial-of-service attack which could be launched by an attacker in order to delay patch distribution.

Recently, Toyozumi and Kara proposed the use of *predators* [28] as a defensive mechanism to protect networks from worms and viruses. Predators are benevolent mobile programs that replicate and migrate from machine to machine across a network, in a manner similar to a virus or worm, disinfecting and immunizing each visited machine.Since predators spread across the network in a tree-like fashion, the patch time will be $O(log_k N)$, where N is the number of clients, and k is the fan-out factor of the predator.

Furthermore, predators have the potential of containing the spread of a virus/worm before a patch is available, by continually disinfecting machines until a stable infection rate is reached. Our results show that, with proper predator design, the stable infection rate can usually be kept to a small percentage of machines without excessive bandwidth consumption. This would serve as a useful stopgap measure for containment of worms/viruses until a patch becomes available. The patch could then be distributed by traditional means or by a second predator.

One issue involving the use of predators is whether or not the predator exploits the existing vulnerability in order to gain access to the system. In the original work on predators [28, 20, 17], this has been suggested as a technique. Intruding into the systems of others without authorization is not legal, and it seems likely that for predators to become a practical technique, they should rely upon OS infrastructure rather than attempt to exploit vulnerabilities to enter systems. More discussion on this topic is in Section 6.

One of the problems that will be faced in downloading any mobile code using either predators propagation, server-push or client-pull technologies is that of authentication of the server and establishing a trusted source. The problem of source authentication is only compounded in the case of a predator, since it is an arbitrary, self-propagating code. The problem can be solved if the vendor supplying the predator signs the code, which can be verified by the recipient. In our work, we have mainly concentrated on studying the effects of a predator once it is accepted by a user machine, and how it could patch and cleanup a network.

The base assumptions underlying our work are as follows. There is a network which is fully connected (any machine can connect to any other machine), there is a consensus to allow predators on the network, and patches can be made available before worms/viruses have penetrated the entire network. Given these assumptions, our work builds on the results of [28] in several ways.

First, in [28], the behavior of predators is predicted using equations from epidemiology [19]. We have developed a discrete simulation tool which simulates the behavior of predators and viruses on small scale (1600 node) networks. We report on the simulation results and closely they match the results predicted in [28].

Second, an important issue is the amount of network traffic generated by the predator. The simulations show that it is possible to design predators so that worms and viruses are eliminated without clogging the network.

Third, the simulation tool is valuable as a design tool for predators, as it allows the predator's behavior to be tuned so that the predator is effective without overloading the network. If predators are to become a practical technique, then tools will be required which allow predator behavior to be confidently predicted before the predator is actually released. We report on the design and implementation of our tool, which is a first step in this direction.

Finally, in [28], in order to conform with biological models, a predator immediately dies when it propagates to an uninfected host. In this paper, we extend the original predator model in three different directions:

- We propose *persistent predators*, which improve the original predator model by introducing a delay before the predator dies.
- We propose *immunizing predators*, which have the ability to install patches as well as disinfect machines.
- We propose *seeking predators*, which have the ability to follow the same path as the virus from an infected machine. This type of predator model is applicable in particular to e-mail viruses, where it may be possible to automatically inspect the mail server log and determine where viruses were sent to and came from.

Our simulations show that the above three predator models are more effective at combating automated attacks without overloading the network.

The rest of this paper is organized as follows. In Section 2, the simulation design, including the predator and virus models, is presented. Section 3 describes the experiments that were performed and the experimental results. Section 4 discusses the implication of the experimental results on the design of predators. Section 5 summarizes related work, and Section 6 discusses broader issues involving the use of predators. Section 7 summarizes our results.

2. Simulation Design

In order to study the propagation of malicious mobile code through a network, we developed a single machine simulation testbed where the application of predators against different types of self-propagating malware could be studied. The main advantages of such an approach over using a real network are ease of configuration and low testbed cost. The obvious disadvantage is that the simulation must be carefully designed in order to accurately model real-world behavior.

The overall simulation works as follows. The network consists of a fixed set of fully interconnected nodes (any

routers that may connect nodes are ignored). Each node has a single e-mail queue and a single user, modeled as a Poisson process, who randomly sends an e-mail (once every 5 minutes on average) or reads all queued e-mails (once every 10 minutes on average). When not performing e-mail functions, users are in an idle state. The e-mail queue length during the simulation is unbounded, although queues which exceed a length of 500 trigger an alert that the e-mail server is overloaded. During the simulations reported on in this paper, no overloading occurred.

At the start of the simulation, all e-mail queues are empty. After the e-mail queues stabilize (which occurs after about 25 minutes of simulated time), a virus-carrying e-mail is introduced into the system. The virus is allowed to propagate until a threshold of infected systems is reached. Once the infection threshold is reached, the predator is introduced to the network. The threshold was varied in order to explore the effectiveness of predators at disinfecting networks in different levels of virus infection. The simulation then runs until some termination criteria is satisfied (e.g., less than 1% of the network is infected).

The simulation uses discrete time, where each cycle of simulation was chosen to correspond to roughly 0.2 seconds. This is a rather arbitrary number — our main concern in this context was to choose a small enough granularity so that the results would be essentially the same as with a simulation based on real time.

The simulation relies on abstract models of predator and worm/virus behavior, which are based on state transitions [18, 24]. The models encode the behavior of a mobile program on a network as a timed finite state machine. Model behavior is tuned by a set of parameters.

2.1. Predator Models

Two basic predator models were employed. The first model, depicted in Figure 1(a), is essentially the same as the model used in [28]. The second model, depicted in Figure 1(b), is an extension which can not only destroy a worm, but can also deliver patches to an infected machine, thereby immunizing the machine from future recurrences of the same attack.

The purpose of the non-immunizing predator is to combat a worm or virus for which no patch is yet available. The non-immunizing predator has the capability to disinfect machines, but not to immunize them from future attacks. In order to prevent the predator from flooding the network, the non-immunizing predator is prevented from propagating unless it finds a copy of the virus. When the predator propagates to a machine which is not infected by the virus, it waits for some time for the machine to be attacked. If no worm or virus attempts to propagate to the machine during the waiting period, the predator dies.

When the immunizing predator (Figure 1(b)) propagates to a new machine, it disinfects the machine if necessary, and then immunizes the machine by applying a security patch regardless of whether or not the machine has ever been infected. The predator then propagates to other machines in the network.

The model states in Figure 1 represent specific phases of the predator's behavior:

- *Initial state*: The predator has just arrived on a machine in the network. The first event that occurs during the initial state is a delay which simulates the transmission and execution time required by the predator. Once the delay is finished, then the predator checks to see if a virus/worm is present on the machine. If a virus is detected, the predator moves to the disinfection state. If no virus is present, then the behavior is dependent on the type of predator: a non-immunizing predator enters the waiting state, while an immunizing predator enters the immunization phrase.
- *Waiting state* (non-immunizing predator only): The predator waits for the machine to become infected. If the predator's time-to-live clock expires before any infection attempt occurs, the predator dies (i.e., it is deactivated and deleted). If the machine is infected by the virus, then the predator proceeds to the disinfection state.
- *Disinfection state*: The predator removes any malware, backdoors, virus copies etc. installed on the machine. The next state is either the immunization state (in the case of an immunizing predator), or the replication state (in the case of a non-immunizing predator).
- *Immunization state* (immunizing predator only): The predator immunizes the machine, by installing relevant security patches. The predator then proceeds to the replication state.
- *Replication state*: The predator generates a number of copies of itself, and sends itself out as an attachment in an email to various users on the network. The number of users are randomly generated, and are generally more than the fan-out of the virus.

2.1.1. Predator Model Parameters In addition to the predator behavioral state transition system, stochastic and temporal aspects of predator behavior are determined by a set of parameters. The parameters are fixed prior to the start of the simulation and apply to all predators. The most important parameters are:

- *Fanout*. The fanout determines how many copies of a self-propagating program are made each time it reaches a new node. The fanout parameters are upper and lower bounds, and the fanout for each propagation is chosen randomly (using a uniform distribution) between the bounds.

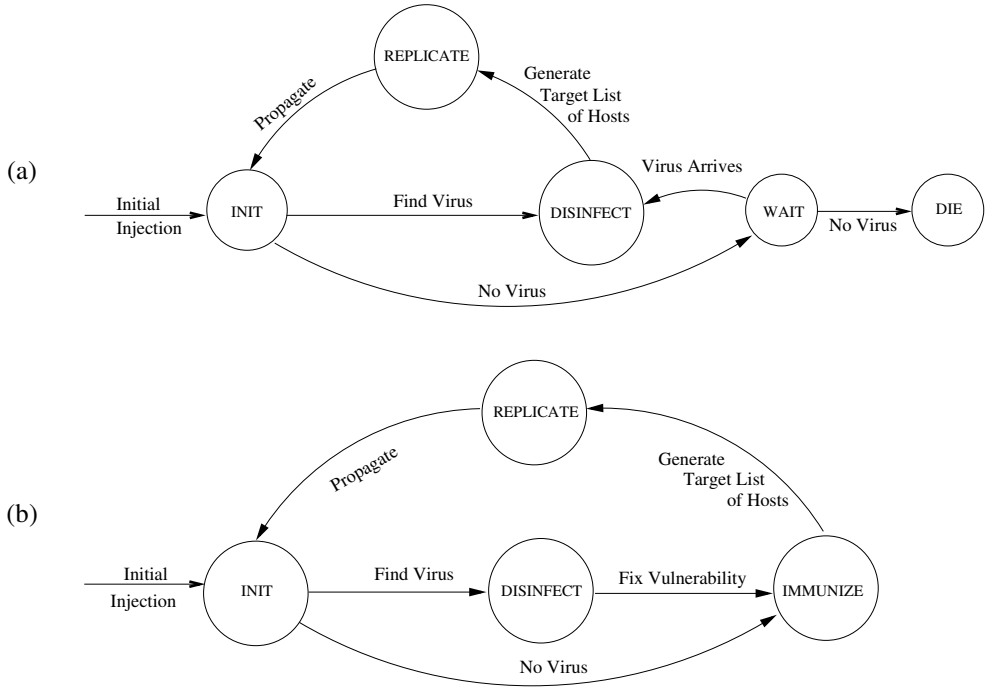

Figure 1. (a) Non-immunizing and (b) immunizing predator models.

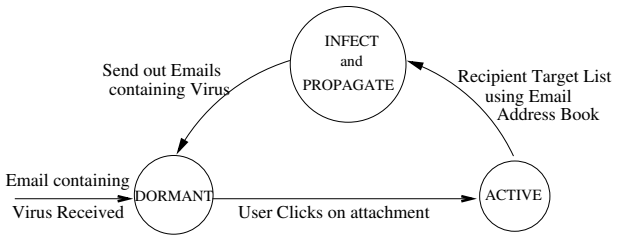

Figure 2. Virus model.

- *Propagation Time*. The delay between the time a predator is sent to a new node and the time at which it is activated. This delay is currently the same for all predator propagation attempts.
- *Time-to-Live*. For non-immunizing predators, the length of time that the predator will wait for a virus to arrive before terminating.

2.2. Virus Model

The virus model used in the simulations appears in Figure 2.2. The virus model has three basic states:

- *Dormant*. The virus is enqueued in the e-mail queue for some node, waiting for the user to activate the virus. The next time the user reads e-mail, the virus will be activated and enter the active state.
- *Active*. The virus accesses the users e-mail history, and randomly chooses some nodes from the history as propagation targets. The number of targets chosen is based on the fan-out of the virus, which is one of the simulation parameters (4 in most of the simulations).
- *Infect and Propagate*. For each selected target user, a copy of the virus is placed in the users e-mail queue. Each of these new virus copies is initially in the dormant state. The original virus then dies.

2.2.1. Virus Model Parameters Just like predators, the behavior of viruses is tuned by a set of parameters which determine the temporal and stochastic aspects. The most important of these are:

- *Fanout*. The fanout is expressed as a uniform distribution with an upper and lower bound.
- *Incubation Time*. The delay between the time a user reads an e-mail containing a virus, and the time the virus becomes active. This delay is also expressed as upper and lower bounds, with the actual incubation time being chosen uniformly randomly between these two values for each virus attack attempt.

These parameters are sufficient to model both flash and stealth viruses. Stealth viruses can be modeled by using a very long incubation period, so that at any given time instant, most of these viruses are in a dormant state, plus keeping the fanout low, so that the percentage of virus-carrying e-mails is kept very low. Flash viruses are characterized by a high fanout and a short incubation time [27, 30].

2.3. Simulation Testbed

The simulator was implemented in Java. With 400 clients, about 800 frequency distributions were maintained, each over 8 time scales. Due to these structures, the total memory use of the Java program was 30MB. When run on a Intel Pentium III system operating at 1GHz running RedHat Linux 9.0, Java2 SDK 1.4, it was able to simulate about 500 cycles per second, i.e., simulate 100 seconds per one second of real time.

3. Experimentation

Our experiments fall into three categories. The first batch of experiments were designed to compare the behavior of the simulator results with the models presented in [28]. The second batch of experiments were designed to test the classic predator model and the effect of various improvements on the model, such as immunization. The effects of varying some parameters, such as the predator fanout and time-to-live parameter, was also studied. The third set of experiments were done to explore the effectiveness of predators against rapidly-spreading worms.

3.1. Simulator vs. Other Models

As pointed out in [28], the interaction between viruses and predators can be modeled using the Lotka-Volterra equations from biomathematics. Let $x(t)$ be the virus population and $y(t)$ be the predator population at any given time instant t. Then, the following differential equations model the virus-predator interactions:

$$\frac{dx(t)}{dt} = r \cdot x(t) - a \cdot x(t) \cdot y(t)$$

$$\frac{dy(t)}{dt} = b \cdot x(t) \cdot y(t)$$

$$(x(0), y(0)) = (x_0, y_0)$$

where x_0 is the initial number of viruses, y_0 is the initial number of predators, r is the viral multiplication rate, a is the predatory rate and b is the predator multiplication rate. The equations have no analytical solution, but are instead approximated numerically. Figure 3.1 shows the result of an orbit generated by a single run of the simulator with a virus fanout of 4 and a predator fanout between 0 and 8, compared with the results predicted by the Lotka-Volterra equations with $r = 1.3$, $a = 0.015$ and $b = 0.015$. In both simulator and the Lotka-Volterra system, $x_0 = 200$ and $y_0 = 1$. The results show that the simulator behaves as a Lotka-Volterra system.

3.2. Predator Effectiveness

In our experiments, we simulated four types of predator:

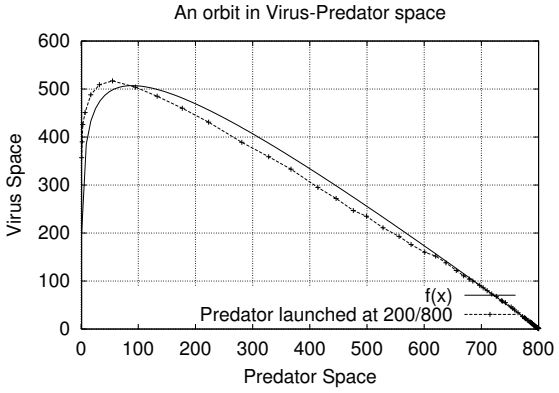

Figure 3. Lotka-Volterra orbit f(x) **compared with simulation results.**

- *Classic predator.* This predator can only disinfect, not immunize, and dies immediately when it propagates to an uninfected system.
- *Persistent predator.* This predator is similar to the classic model, the only difference being that it doesn't die immediately when an uninfected system is encountered. Instead, it waits for a fixed amount of time, during which it destroys the first incoming virus and propagates, or dies if no virus arrives.
- *Immunizing predator.* This predator differs from the previous models in two ways. First, whenever it reaches a new system, it renders that system permanently immune to any future attacks. Second, it propagates regardless of whether a virus is found on the system or not.
- *Seeking predator.* This predator is an extension of the immunizing predator, with the ability to inspect e-mail logs and follow the same path as an incoming or outgoing viruses. This enables the predator to target infected machines rather than propagating randomly.

Figure 4 compares the effect of the classic and immunizing predators against an e-mail virus. In these experiments, the network size was 800, virus fanout 4, and predator fanout 8. In every case, the predator was injected into the network when the infection rate reached 25%. As the results show, the immunizing predator is superior to the classic, and manages to disinfect the network at a rate roughly the same as the rate at which it was infected. Nevertheless, it should be pointed out that the classic (and persistent) predator is still useful for combating a virus or worm for which no patch is available, or in situations where the use of an immunizing predator is not feasible.

Figure 5 shows the effectiveness of increasing the fanout of the immunizing predator. When the fanout is increased

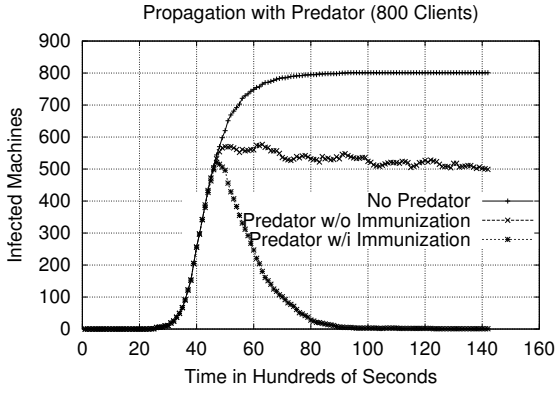

Figure 4. Predator effect on virus population.

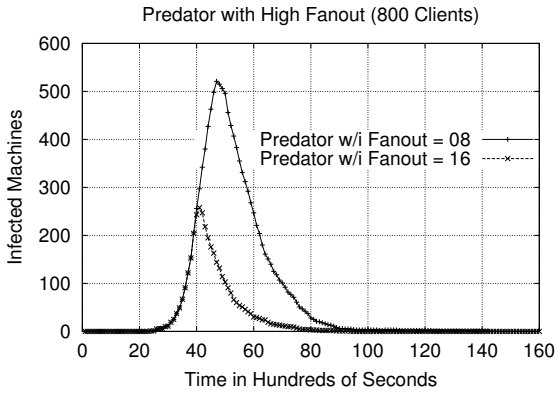

Figure 5. Fanout effect on performance.

to this amount, it has the effect of immediately arresting the spread of the virus and quickly cleaning up the network.

An additional and perhaps surprising benefit of increasing the fanout is that the overall network traffic level is reduced. Figure 6 shows observed virus and predator traffic The normal email traffic before the virus hits the network is around 8-9 emails/sec. The virus was introduced in the system at t=2898 secs, at which point one can clearly see an exponential increase in total email traffic. The predator is introduced at around $t = 4100$ secs, when 200 out 800 machines are infected. The predator traffic never increases beyond 5 emails/sec. The predator has a fanout of 8, but it will die in 75% of the cases initially, since only 200 out of 800 machines are infected, and will effectively propagate with a fanout of only 2. So, effective fanout of the predator will depend on the percentage of infected machines. By the time $t = 9000$ secs, the virus is completely eradicated.

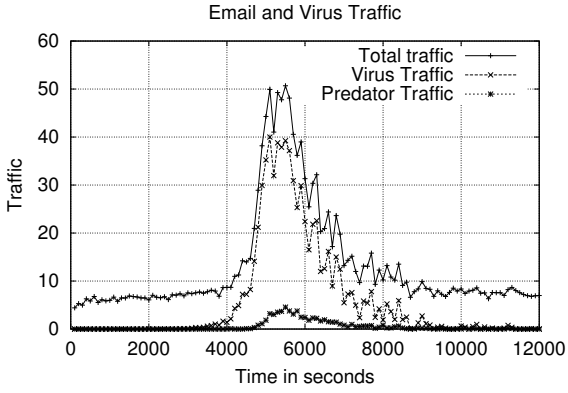

Figure 6. Network traffic when fanout=8.

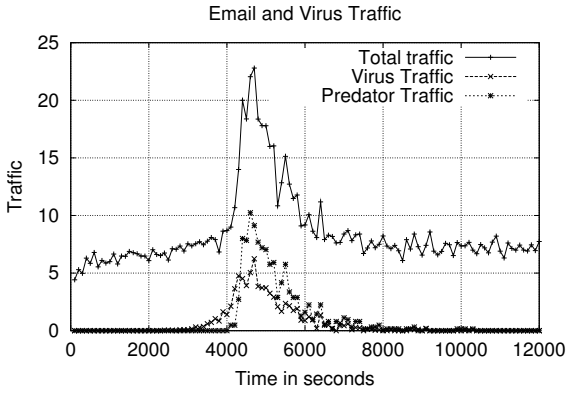

Figure 7. Network traffic when fanout=16.

Figure 7 shows the traffic characteristics when the fanout of the predator is increased to 16, which is 4 times the fanout of the virus. As in the earlier case, virus was introduced at t=2898 secs, and the predator was introduced when 25% of the machines became infected. As the results show, predator traffic peaks much higher than in Figure 6, though the total traffic is still much less, due to quick containment of the virus. By time $t = 7500$ secs, the virus has been completely removed from the system.

The conclusion one can draw from this that the in cases where the size of the predator is close to the size of the virus, then increasing the fanout will eradicate the virus while consuming less network bandwidth. In cases where the size of the predator is much larger than the size of the malware (e.g., a large patch must be installed in order to immunize each machine), then the overall traffic may increase when the fanout is set to 16.

Figure 8 illustrates the effectiveness of the persistent predator. Recall that the persistent predator does not im-

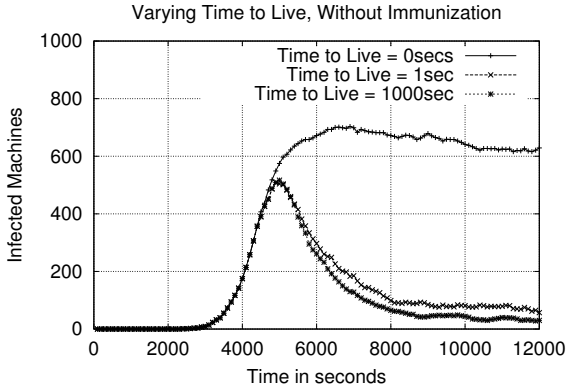

Figure 8. Persistent predator effectiveness.

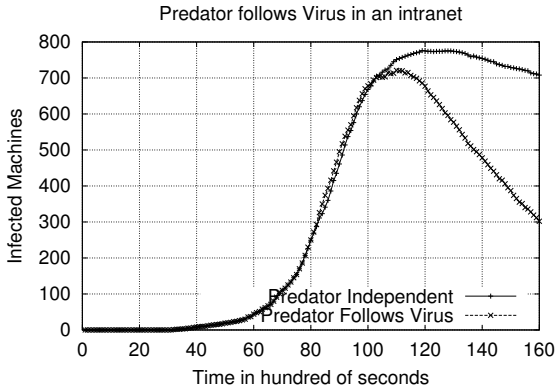

Figure 9. Seeking predator effectiveness.

munize the machine, but is capable of killing the virus as long as it is present on the machine. When the time-to-live is zero (the classic predator), we get a stable system, where virus and predator populations balance out, and the number of infected machines becomes stable at 600 out of 800, which is 75% of the whole network. If we increase the time-to-live to even a small positive value, the stable population of infected machines drops down to less than 5%. Furthermore, the rate of decrease is close to that observed for the immunizing predator. Of course, there are still a number of major drawbacks in comparison to an immunizing predator, such as the fact that at any point in time a small but significant number of machines are infected by the virus, so that the predator and virus will continue to consume network and processing resources ad infinitum.

Figure 9 illustrates the effectiveness of the seeking predator, which is nearly identical to the immunizing predator, with the enhancement that it has the ability to perform a forensic analysis a determine which hosts the virus

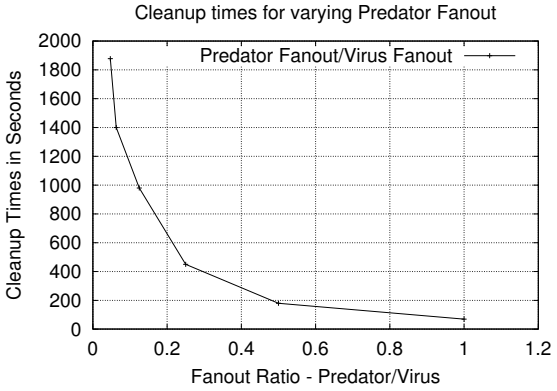

Figure 11. Clean up time as function of predator-to-virus fanout.

or worm is likely to have infected and propagate to those machines first. As Figure 9 shows, the ability to seek significantly improves performance.

3.3. Rapidly-spreading worms

A third set of experiments were done to explore the effectiveness of predators against rapidly spreading worms/viruses. Of particular concern is the fanout of the predator required to quickly eliminate worms from the network, as the designer of a predator wants to use the minimum amount of fanout required to eliminate the virus/worm in a reasonable amount of time.

Along these lines, experiments were done to measure the clean-up times required for varying degrees of predator and virus fanout. The results of these experiments are shown in Figure 10. In these experiments, a rapidly spreading virus was introduced simultaneously with the predator 20 seconds into the simulation (the purpose of the delay was to allow the size of the simulated e-mail queues to stabilize). Note that while this simulation is inspired by flash worms, no simulation of the network congestion likely to be caused by a flash worm was done. An additional difference between this experiment and the earlier ones is that, in this experiment, $\frac{1}{8}$-th of the machines were vulnerable to the virus, while in the earlier experiments, 100% of the machines were vulnerable.

Based on these experiments, the correlation between the ratio of predator fanout to virus fanout and the clean-up time was measured. The results, shown in Figure 11, indicate that a predator fanout of $\frac{1}{2}$ the virus fanout is sufficient to quickly eliminate viruses from the network under the simulated conditions. This is an encouraging result, as as a fanout ratio of less than one means that the amount of

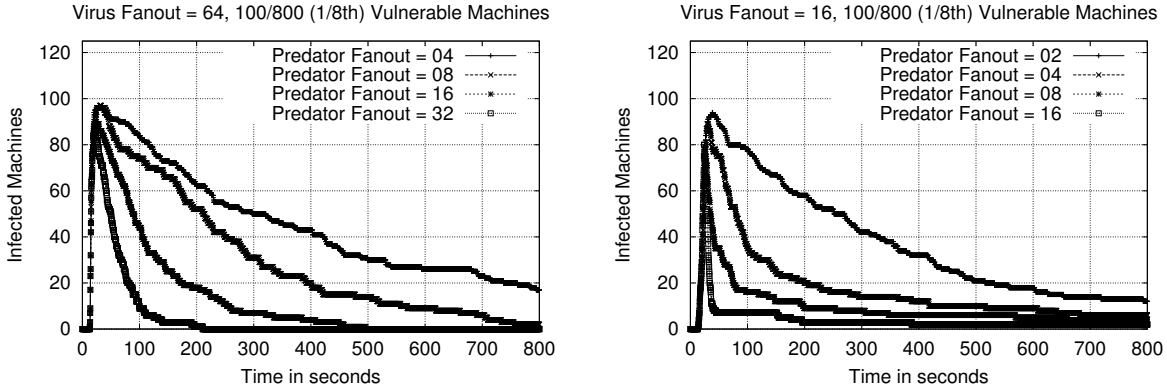

Figure 10. Effect of fanout on cleanup times for rapidly-spreading viruses.

predator traffic is likely to be less than the virus traffic.

4. Predator Design

When designing a predator, the goal is to minimize the number of machines which become infected and disinfect infected machines as quickly as possible, while keeping performance overhead to a minimum, and minimizing the risks introduced by the predator. To reach these goals, the designer must choose between several competing alternatives.

The first choice is "should the predator be immunizing or non-immunizing?" A non-immunizing (i.e., non-patching) predator could be used in cases where no patch is yet available, the risks involved in patched are judged to be too great, or a predator-driven application of the patch would cause too much disruption to ongoing computations.

In these cases, a non-immunizing predator could be used to reduce the number of infected machines to very low level until patches are available and/or manually applied to each machine. A second possibility for dealing with novel attacks is a two-phase use of predators. In the first phase, a non-immunizing predator contains the spread of the new attack, and in the second phase, an immunizing predator patches all vulnerable systems.

A second choice is whether or not the predator has the seeking property or not (i.e., whether or not it follows the path of the virus). This choice is driven primarily by whether or not the outgoing path of the virus/worm can be determined, and if so, how much effort is required. While a seeking predator enjoys a significant performance advantage over a non-seeking one, the extra implementation effort may not be worthwhile, particularly for non-e-mail worms.

Additional choices include the setting of the predator parameters, namely the time-to-live and fanout. These involve tradeoffs between performance overhead and effectiveness, and are also driven by the properties of the virus.

The fanout value depends primarily on the type of attack. For slowly propagating worms, a predator fanout equal to twice the worm/virus fanout was shown to be effective (see Figure 5), which is acceptable because the virus in used in those experiments was spreading at a much lower rate, so that the overall network traffic (shown in Figure 7) remained reasonably low. For rapidly propagating attacks, a predator fanout in the range of 1/5 to 1/2 of the attack fanout appears to be effective (see Figure 10).

The time-to-live value depends on the type of predator. For immunizing predators, the time-to-live is irrelevant; for all non-immunizing predators, a large time-to-live is desirable, and ideally would be on the order of the time required for the predator to reach every node on the network, but will consume resources on each protected machine.

5. Related Work

Work related to ours falls into three essential areas:

Predator-based approaches. The earliest works which suggest the use of predators are [20, 17]. [28] is the seminal paper which suggests the use of models from mathematical biology to predict the behavior of predators. Our work extends theirs by extending the range of potential behaviors to patch management as well as virus removal, and using discrete simulations to study predator behavior.

Patch management and distribution techniques. In reaction to the escalating security situation, some operating system vendors, such as [23] are attempting to automate the distribution and application of security patches. The main difference between these efforts and the predator approach is that the current OS vendor approaches are largely centralized, which suffers from distribution bottlenecks that the predator approach avoids. There has also been investigation into the use of mobile agents to perform software updates, such as [1]. The focus of these works has been the develop-

ment of infrastructure, rather than studying the likely effectiveness, which is the main focus of our work.

Worm modeling techniques. There has been one early [21] and several recent [29, 13, 15, 16] efforts to model worm behavior. The main difference between these approaches and ours is that they focus on the spread of the worm itself (typically modeling cleanup as a simple algorithmic parameter), while this paper focuses on a specific technique to combat the spread (i.e., predators). [31] employed simulations to explore the use of quarantines to stop the spread of worms. [22] used a detailed simulation to evaluate the effectiveness of worm detection algorithms. [3] used simulations to compare the ability of various scale-free network topologies to preserve critical functionality in the presence of self-propagating attacks.

6. Discussion

While we are encouraged by our results, which we believe indicate that predators have the potential to be developed into a practical approach for combating worms and viruses, there are a number of issues involving the use of predators which warrant further discussion.

6.1. How the predator gains entry

One issue confronted by a person wishing to release a predator onto a network is legality. If the predator exploits the same vulnerability as the virus, as was originally suggested, and not all users on the network have given consent, then the release of the predator is a criminal act in many countries. There are two solutions one can envision to this problem. First, there may eventually be some sort of legal authority which authorizes the release of predators, perhaps similar to today's public health agencies which have the power to quarantine infectious individuals Second, a *predator port* infrastructure could support the entry of authorized predators. The first approach has the advantages that no new infrastructure is required, and no new vulnerabilities are created, with the drawback being the virus or worm could potentially close the vulnerability after gaining entry, preventing the predator from removing the virus or patching the system. The second approach requires significant implementation effort, and must be done carefully to prevent unauthorized access, but has the advantage that, if implemented well, the malicious code will be unable to close the predator port, and propagation of the predator will be easier to control.

6.2. Techniques for secure patch distribution

Distributing code through predators poses security challenges similar to those faced when mobile code is downloaded over a network. In particular, if each system on the network has an automated predator port enabled, then the potential exists for an unauthorized predator to subvert every system on the network. Hence, ensuring the integrity and authenticity of predators is essential. Towards these ends, the following techniques can be used:

Transit Integrity: To verify that the patch was not damaged in transit, a cryptographic hash can be transmitted in addition to the patch code. The hash value can be locally verified to ensure that the patch was transmitted correctly.

Digital Certificates. Code-signing certificates can be used to authenticate the predator.

Centralized authentication. Upon receiving a predator, the system could query a known centralized server to check the authenticity of the predator. The size of the authentication query would hopefully be much smaller than the size of the patch, thereby avoiding bottleneck issues.

6.3. Some risks with patch management

Software patches implicitly contain vulnerability information which may be abused to jeopardize the security of a system. Malicious users can analyze patches and develop exploits against unpatched systems. These risks can be mitigated by (a) rapidly distributing patches so that all systems are patched within a small timeframe, and (b) encrypting patches in ways which prevent reverse engineering.

Unfortunately, neither of these solutions appears to be feasible at the moment. On the positive side, it should also be noted that predators can be useful even in cases where no patching is done by the predator, so that if the risks are judged to be too high, a non-patching predator could be used to contain a virus/worm outbreak until a patch could be distributed though some other distribution channels.

6.4. Simulation Issues

The main weakness of the results presented in this report is that they are all based on simulation. Real systems often display behaviors that are more complex and variable than those exhibited in simulations. In order to truly assess the effectiveness of the predator approach, it will be necessary to evaluate it using realistic network traffic.

A second difficulty in extrapolating the simulation results is that on real systems, network-based application traffic crosses organization boundaries frequently. For example, an email virus may propagate from one user to any other user on the Internet, and not just on the intranet of the user's organization. The effects of firewalls, routers and network topology on predators have not been accounted for. These issues need to be addressed in future research.

7. Conclusion

The results presented in this paper demonstrate that predators have the potential to quickly clean-up networks

infected by self-propagating malicious code and also immunize networks from future attacks. Predators have a potential for becoming a practical emergency patch distribution mechanism, when many machines need to be quickly patched in the face new a worm or virus. Simulation techniques could be used to tune the predator's behavior prior to release, so that worms are quickly eliminated while the only minimum amount of necessary bandwidth is consumed. Predators can potentially provide timely control on the spread of self-propagating worms, thereby reducing the monetary losses due to their unchecked spread.

References

[1] L. Bettini, R. D. Nicola, and M. Loreti. Software update via mobile agent based programming. In *Proceedings of the 2002 ACM symposium on Applied computing*, pages 32–36. ACM Press, 2002.

[2] S. Bhatkar, D. C. DuVarney, and R. Sekar. Address obfuscation: an efficient approach to combat a broad range of memory error exploits. In *USENIX Security Symposium*, 2003.

[3] L. Briesemeister, P. lincoln, and P. Porras. Epidemic profiles and defense of scale-free networks. In *ACM Workshop on Rapid Malcode*, 2003.

[4] Cert advisory ca-1999-04 melissa macro virus. http://www.cert.org/advisories/ca-1999-04.html.

[5] Cert advisory ca-2001-22 w32/sircam malicious code. http://www.cert.org/advisories/ca-2001-22.html.

[6] Cert advisory ca-2001-26 nimda worm. http://www.cert.org/advisories/ca-2001-26.html.

[7] Cert advisory ca-2003-04 ms-sql server worm. http://www.cert.org/advisories/CA-2003-04.html.

[8] Cert advisory ca-2003-04 w32/mydoom.b virus. http://www.us-cert.gov/cas/techalerts/TA04-028A.html.

[9] Cert advisory ca-2003-20 w32/blaster worm. http://www.cert.org/advisories/CA-2003-20.html.

[10] Cert advisory ca-2004-04 email-borne viruses. http://www.cert.org/advisories/CA-2004-02.html.

[11] Cert. code red ii: Another worm exploiting buffer overflow in iis indexing service dll. http://www.cert.org/incident_notes/in-2001-09.html.

[12] Cert incident note in-2003-03 w32/sobig.f worm. http://www.cert.org/incident_notes/IN-2003-03.html.

[13] Z. Chen, L. Gao, and K. Kwait. Modeling the spread of active worms. In *IEEE Infocom*, 2003.

[14] S. Forrest, A. Somayaji, and D. H. Ackley. Building diverse computer systems. In *Workshop on Hot Topics in Operating Systems*, 1997.

[15] S. Gorman, R. Kulkarni, L. Schintler, and R. Stough. A network based simulation approach to cybersecurity policy. http://policy.gmu.edu/imp/research.html.

[16] S. P. Gorman, R. G. Kulkarni, L. A. Schintler, and R. R. Stough. A predator prey approach to the network structure of cyberspace. In *Proceedings of the winter international synposium on Information and communication technologies*, pages 1–6. Trinity College Dublin, 2004.

[17] R. Grimes. *Malicious Code*. O'Reilly and Associates, 2001.

[18] A. Gupta and R. Sekar. An approach for detecting self-propagating email using anomaly detection. In *Recent Advances in Intrusion Detection*, 2003.

[19] J. Jorgensen, P. Rossignol, M. Takikawa, and D. Upper. Cyber ecology: Looking to ecology for insights into information assurance. In *DARPA Information Suvivability Conference and Exposition*, 2001.

[20] A. Kara. On the use of intrusion technologies to distribute non-malicious programs to vulnerable computers. Technical report, University of Aizu, 2001.

[21] J. Kephart and S. White. Directed-graph epidemiological models of computer viruses. In *IEEE Computer Society Symposium on Research in Security and Privacy*, pages 343–359, 1991.

[22] M. Liljenstam, D. M. Nicol, V. H. Berk, and R. S. Gray. Simulating realistic network worm traffic for worm warning system design and testing. In *Proceedings of the 2003 ACM workshop on Rapid Malcode*, pages 24–33. ACM Press, 2003.

[23] Patch management, security updates, and downloads. http://www.microsoft.com/technet/default.mspx.

[24] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou. Specification-based anomaly detection: a new approach for detecting network intrusions. In *ACM Computer and Communication Security Conference*, 2002.

[25] S. Sidiroglu and A. D. Keromytis. Countering network worms through automatch patch generation. Technical Report 029-03, Columbia University Department of Computer Science, 2003.

[26] S. Staniford. Analysis of spread of july infestation of the code red worm. http://www.silicondefense.com/cr/july.html.

[27] S. Staniford, V. Paxson, and N. Weaver. How to own the internet in your spare time. In *Usenix Security Symposium*, 2002.

[28] H. Toyoizumi and A. Kara. Predators: Good will mobile codes combat against computer viruses. In *New Security Paradigms Workshop*, 2002.

[29] Y. Wang and C. Wang. Modeling the effects of timing parameters on virus propagation. In *ACM Workshop on Rapid Malcode*, 2003.

[30] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham. A taxonomy of computer worms. In *ACM Workshop on Rapid Malcode*, 2003.

[31] C. Zou, L. Gao, W. Gong, and D. Towsley. Monitoring and early warning for internet worms. In *ACM Computer and Communication Security Conference*, 2003.

[32] C. Zou, W. Gong, and D. Towsley. Code red worm propagation modeling and analysis. In *ACM Computer and Communication Security Conference*, 2002.