

Hierarchically Controlled Co-operative Response Strategies for Internet Scale Attacks

C.G.Senthilkumar
Department of Computer Science
University of California, Davis

Karl Levitt
Department of Computer Science
University of California, Davis

1/5/03

Abstract

Responding to Internet scale attack poses many new problems. First is the wide spread nature of the victims across the Internet. Individual organizations trying to manage attacks locally using only local knowledge and resources will have little effect on Internet scale attacks like worms. Response mechanisms must necessarily extend beyond the borders of any single organization. The second problem is the way different inter-organization control structures perform when responding to Internet scale attacks. Our previous studies have shown that peer-to-peer cooperative control gives widespread alarm coverage to permit distributed response, but propagation of the necessary information is slow. Hierarchical control may produce a faster mitigating response to a detected spreading attack. In this paper we study hierarchically arranged cooperative control structures as a strategy for responding collectively to a large-scale Internet worm. We use a simulated hierarchical control structure to evaluate response strategies based both upon their ability to block malicious worms, fast and slow spreading, and to tolerate ambient false alarm environments.

1 Introduction

The importance of networked computers and their security in light of recent events related to terrorism is unprecedented. There is no need to belabor the potential havoc that a malicious hand can fix on our lives, upon gaining access to critical computer installations. Critical installations include computer sys-

tems of the world nations' defense forces, power systems, the Internet backbone routers, domain name servers, etc.,. The importance of securing such infrastructure from being compromised or misused can never be over-emphasized.

One of the various ways in which computer systems can be compromised is by deploying a computer worm¹. There have been instances in the past where such worms have virtually brought the Internet to a grinding halt such as the "Internet worm".

This paper discusses the history, evolution and some of the latest models of highly virulent computer worms in section 2. Section 3 talks about the various aspects of worm defense. The next 2 sections are devoted to the models of worm defense proposed by the UC Davis security lab. The "Friends Model" of worm defense is described in section 4. The focus of the paper, "Hierarchically Controlled Co-operative Response Strategies" is discussed in section 5. The simulation of the protocol, the results and the future direction of the research are dealt with in detail in the remaining parts of the paper.

2 Background

Computer worms, as they are widely referred, were first reported by Shoch and Hupp at the Xerox Palo Alto Research Center(PARC) in the early 1980's after much ground work starting in the early 1970's. The worm programs were an experiment in the devel-

¹Numerous word wars have been fought to define a worm. In this discussion we will consider a worm as any computer program which once started in a networked host can autonomously find and run on other machines.

opment of distributed computations: programs that span machine boundaries and also replicate themselves in idle machines. They were quite successfully used to support several real-time applications. An apparent corruption in the worm code during migration leaving about 100 machines dead in PARC helped show the dangerous potentials of a worm.[11]

In the evening of November 2, 1988, the *Morris worm* was released upon the Internet which invaded VAX and Sun-3 computers running versions of Berkeley UNIX. Within the space of hours this worm spread across the U.S, infecting thousands of computers and making many of them unusable due to the performance drain of the worm. The mechanics of the worm makes a very interesting study[12, 14, 7]. The worm also employed methods to cover its trails and adopted camouflage to evade detection. This was the first major incident which made computer scientists around the world to sit up and take a really serious note of worms[10, 13].

Of the more recent worms, *Code Red I, II* and *Nimda* were the most reported. *Code Red* exploited a Microsoft IIS Web servers' vulnerability and is still active re-surfing monthly, while *Nimda* did the same in addition to looking out for the back doors left behind by *Code Red II* and *Sadmind* worms.

The main difference amongst them was that Code Red I generated random IP addresses to infect while the other two used *local sub-net scanning* which was more effective.

2.1 Network Scanning Techniques

The spread of active worms is generally limited by how quickly new potential hosts can be discovered[16]. [16] discuss several scanning strategies like *hit-list scanning*, *permutation scanning*, *topological scanning*, *local sub-net scanning* and *Internet-sized hit-lists* which can stupendously increase a worm's virulence.

Topological scanning, a technique that uses information contained on the victim machine to select new targets, has been used by *e-mail viruses* and also by the *Morris worm*. *Local sub-net scanning* has been used by *Code Red II* and the *Nimda* worms.[16].

In a *permutation scan*, all worms share a common pseudo random permutation of the IP address space.

Any machine infected during the hit-list[16] phase or local sub-net scanning starts scanning just after their point in the permutation and scan through the permutation, looking for vulnerable machines. Whenever it sees an already infected machine, it chooses a new, random start point and proceeds from there. Worms infected by permutation scanning would start at a random point.

This has the effect of providing a semi-coordinated, comprehensive scan while maintaining the benefits of random probing and minimizing duplication of effort. After any particular copy of the worm sees several infected machines without discovering new vulnerable targets, the worm assumes that effectively complete infection has occurred and stops the scanning process. A timer could then induce the worms to wake up, change the permutation key to the next one in a pre-specified sequence, and begin scanning through the new permutation [16].

2.2 Warhol and Flash Worms

2.2.1 Warhol Worms

Before a worm is released, the worm author could collect a list, say, 10,000 to 50,000, of potentially vulnerable machines, ideally ones with good network connections. The worm, when released onto a machine on this list, the *hit-list*, begins infecting down the list. When it infects a machine, it divides the hit-list into half, communicating one half to the recipient worm and keeping the other half. [16] calls such a worm, *Warhol Worm* and such a scanning technique, *hit-list scanning*. It also talks about distributed control and update mechanisms by which such worms can be maintained and controlled on the Internet.

2.2.2 Flash Worms

An improvised Warhol strategy would be to program the worm to divide the *hit-list* into 'n' small blocks instead of 2 huge ones, infect an especially high-bandwidth address in each block and pass on to the child worm the corresponding block of the *hit-list*. This process would be repeated by the each child worm.

A threaded worm could start infecting hosts before it had received the full host list from its parent to work

on; to maximize the parallelization process, and it can also start looking for multiple children in parallel. [16] discusses more sophistications and concerns of this strategy.

[16] argues that it is reasonable to think that a determined attacker could have built a list of all or almost all servers with the relevant service open to the Internet in advance of the release of the worm. There are about 12M web servers on the Internet and so, an address list will be 48MB which can be pushed down a T1 line in about 4 minutes and through a OC12 link in under a second.

Such a worm, with an *Internet-sized hit-list*, starting from a site with high bandwidth links, can in principle infect all of the vulnerable machines on the Internet in under about 30 seconds and is termed a *Flash Worm*[16].

2.3 Stealth Worms

Though *Flash Worms* can attack with lightning speed such that no human mediated efforts could be of any use, we could still devise automatic means of detecting and stopping them. But *stealth worms* are of a class that spread much slowly, evoke no peculiar communication patterns and spread in a contagion fashion.[16]

2.4 Polymorphic Worms

Any worm that changes its form or functionality as it propagates from machine to machine can be called a *Polymorphic Worm*, though there is no clear cut definition for it as yet.[9, 19]

2.5 Miscellaneous Worms and viruses

Some of the second generation viruses include *Retro viruses* which fight back against anti-virus tools by deleting virus definition tables, memory resident scanners, etc., *Stubborn viruses* that can prevent themselves from being unloaded from an infected Windows system and *Wireless Worms* that can infect wireless devices by making use of their ability to exchange applications “through the air”.[8]

3 Aspects of Automatic Mitigation

3.1 Detection vs. Declaration

A known worm’s presence in a machine could be *detected* fairly easily. For eg., the presence of a certain string of characters in the log file of a server indicates that a *Code Red* worm attempted to infect that server[5]. There are several commercial tools available to detect any unusual conditions on the network and also to find and cure known worms and viruses. But there is no tool that can detect an anomaly and declare with certainty that it is a worm spreading.

“How to *declare* a worm?” is a big question. Whenever there is any abnormal activity on the network, it could be a worm, DDoS, a patch update to applications or just an innocent vagary of the Internet. The situation demands a lot of correlation of activities, symptoms and data at various places to determine if an anomalous activity on the network is a *worm or not*. The security lab at UC Davis is currently involved in studying this aspect of worm defense.

Fast moving worms are easy to detect because of the sudden anomalous increase in traffic but are difficult to stop. They could potentially take over the entire Internet in a matter of few minutes[16]. Slow moving worms on the other hand are difficult to detect due to lack of obvious symptoms but once detected, they are easy to stop. Any automatic mitigation strategy should be able to address both these kinds of worms.

Also, the defense systems should not over-react to false alarms or diversionary attacks. For example, suppose the defense strategy is to just shut down all services in response to any attack. And further suppose that the goal of an adversary is to shutdown all services, which is a reasonable goal. The adversary doesn’t have to work too hard to spoof an attack and achieve his goal. Even a false alarm would shut down all services. That is to say the defense mechanism should not be fooled into inflicting a self denial of service. An ideal “Stop Worms” strategy would be able to stop even polymorphic worms and worms that exploit unknown vulnerabilities.

Of course, the normal traffic is affected when the alerted routers are scanning for worm-like traffic. But it is only a small price to pay for a healthy Internet.

In case of false positives, it is an unworthy price paid. A false negative is going to be very embarrassing, if not devastating.

3.2 Prevention

“Prevention is always better than cure” applies here too as it does everywhere else. Some of the preventive steps include:

- Having the latest anti-virus s/w, sensible filtering (such as quarantining all executable content from e-mail and Web traffic), fire-walling, and religiously maintaining patches.[17]
- Configuring the firewall properly, “all that is not explicitly allowed is forbidden.”[16, 1]
- Including security by design², sensible defaults^{3,4}, a diversity of OS in the network, usage of type-safe languages to write network services and quicker transition to IPv6. The last one can just slow down the spread but not stop it completely.[18]
- Using techniques, which linguistically prevent buffer overflows, such as safe-C dialects, buffer overflow analysis, non-executable stack and heap policies and Software Fault Isolation⁵ are essential features of any defense: they prevent the common holes from developing.[16]
- Employing Fine grained mandatory access controls. This ensures that every system call is authorized as a function of the running program, the user running the program and the context that invoked the program.[17]
- Evolving a protocol to isolate infected and compromised machines from the Internet.[16]

²UNIX wasn't designed with security as a goal.

³Default DEBUG mode in sendmail in SUN machines was exploited by the Morris Worm.

⁴Microsoft Iis is an amazingly vulnerable target for worms. But is still installed by default with Windows 2000 and it also provides a highly homogeneous target

⁵This technique creates a Java-like sandbox for dynamically loading arbitrary code in a language- neutral manner.

3.3 Mitigation

The obvious goal of any mitigation mechanism is to stop the spread of a worm. But it is not easy to stop all instances of the worm all at once, once it is detected at one place. However, we can hope to confine its spread to a certain percentage of all the vulnerable hosts or to a certain subnet of the Internet.

Upon detection of the spread of a worm, an alert message or a trigger could be sent to a pre-built list of vulnerable, co-operating hosts. This trigger would put in place or invoke the necessary defensive mechanisms in machines that it could reach. The defensive mechanisms would include:

- Closing all unnecessary⁶ out-going connections, so that the chances of worm spreading out from the infected machine is minimized.
- Filtering traffic with the worm signature at *routers* or *border gateways* (BGW) or any such place as deemed appropriate. *More on this point later.*
- Employing a *Sticky Honey-pot* by creating virtual machines using unused IP addresses in the local network[3]. This in effect is a reverse Dos on the worm. This slows the spread of a worm and is an effective use of deception as a counter-offensive.

The “hierarchical model” of worm defense discussed in this paper tries to show that we can confine the worm to a certain given percentage of all the vulnerable hosts by assigning appropriate values to the various variables that make up the model. This model also handles false alarms and stealth worms.

4 Previous work - The Friends Model

The Friends’ model[6] is based on the willing cooperation of a set of participating hosts on a pre-arranged protocol. It also assumes that the worm is not polymorphic.

⁶The site managers or all participating hosts could collectively define “unnecessary”.

Once we know that there is a worm spreading in the Internet, we need to spread an alert message to the set of participating hosts to stop its spread. This alert can be sent from the detector to the entire set or a small subset depending on its ability to reach other hosts and other factors that we'll discuss below.

4.1 The Worm Race

All participating hosts have a peer-to-peer and an hierarchical relationship with other hosts as in the real Internet. For eg., Border Gateways are higher in the hierarchy than routers.

Each participating host has a list of trusted hosts⁷ with its associated trust-worthiness. The trust-worthiness of a host higher in the hierarchy is higher than that of a peer, than that of one lower in the hierarchy. This list need not include all the other participants.

Once a worm is suspected, we send its suspected signature and its associated *perceived threat*, PT , to the other friendly or trusted hosts. But, how a worm is suspected, detected or declared is a big question and has been discussed in the previous section. Such alerts received, the trust-worthiness of the alerting hosts and the actual number of suspicious packets seen by the host form the inputs to calculate the PT .

$$\begin{aligned} \text{Perceived Threat, } PT = & \\ & f(\# \text{ of alerts received,} \\ & \text{trust-worthiness of the alerting hosts,} \\ & \# \text{ of malicious packets seen}) \end{aligned}$$

if($PT > \text{Threshold}$) Take actions.

The hosts that receive the alert can take actions based on some criteria like receiving the same alert from a certain minimum number of hosts with a certain minimum trust worthiness. This way, we don't end up spreading spurious alerts originating from malicious or *already compromised* hosts.

The list of friends is not static. It can be changed on a periodic basis based on the veracity of the previous alerts from each of the them automatically or by

⁷The terms trusted hosts and friends are used interchangeably.

the site administrator. The protocol to be followed while updating the friends lists will be the same as in updating Internet routing tables to avoid race conditions and other inconsistencies that are possible when data at different locations have to be updated independently. Of particular interest would be the situation where one or more of the participants are left out of the friends lists of all the other participants due to race conditions.

The actions taken are decided by the hosts individually unless there is an understanding with other hosts. And the actions taken by each host could include:

1. Assigning a threat level as perceived by the alert-receiver based on various parameters as mentioned above. This PT may be different from the PT as seen by others.
2. Alerting its *friends*. Depending on the PT , a host chooses a different number of *friends* to alert.
3. Scanning the incoming and outgoing packets for the new signature. The intensity of scanning depends on the PT at the host. Based on the results of scanning, the PT at that host might increase or decrease.
4. If the PT changes based on scanning, sending new alerts with the new PT . This acts like a control mechanism to dynamically increase or decrease the scanning intensity.
5. Reducing the bandwidth available to the general traffic and increasing the bandwidth to the alert messages. This is possible because the hosts can control the traffic speed passing through it.
6. Blocking of traffic believed to be malicious.
7. Backing-off from blocking the allegedly malicious traffic once it is found not to be so. This is achieved *automatically* since the intensity of the scanning decreases if there is a decrease in PT .

This spread of the alert messages may also be seen as a worm, but benign. Hence the topic **Worm Race**. One should also note that since we, the defenders, know the targets for our alerts, we can spread the benign worm much faster than the malicious worm after the initial latency of detecting a worm and extracting its signature.

The defending hosts might include backbone switches, routers, border gateways or gateways at universities, etc.,.

The higher in the network hierarchy that we deploy our scanning, the more costly is the process because of the high volume of traffic. But we can stop the worm at a much earlier stage and can also spread the alert message quickly as there are less targets for our alerts.

Note that if the worm already crossed this host and entered the network below it, the hosts in the sub-net are open to attack. So, we need to have the detectors at various levels of the Internet hierarchy not just at one level. For eg., both at border gateways and routers; not just at border gateways. We should also not increase the redundancy too much to inflict a self DOS wherein a majority of the machines end up just scanning the traffic passing through it. We need to choose an optimal redundancy level to avoid such single point failures.

Fig.3 shows the hierarchical relationship among the BGWs, routers and hosts.

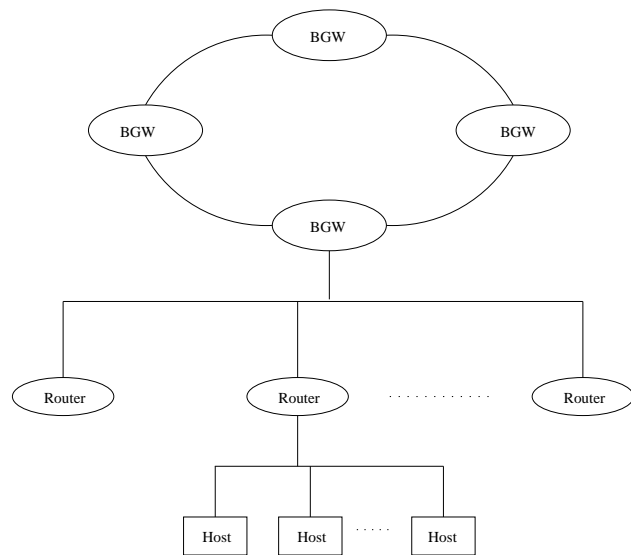


Figure 1: Hierarchical relationship among BGWs, Routers and Hosts. Ordinary hosts are one level below routers which are one level below BGWs.

While scanning of incoming traffic prevents infection, scanning of out-going traffic prevents the spread of the worm and helps in quarantining the infected sub-net.

4.2 Simulation of the Friends Model

The Friends Model was simulated using the SWARM simulator. SWARM is a software package for multi-agent simulation of complex systems[2]. SWARM was chosen for its ease of programming and its active developmental support.

The simulation begins by starting the infection from random hosts and then arbitrarily assigns a host as the initial detector of the infection. This host sends an alert to 8 other randomly chosen hosts. This alert, the *white worm*,⁸ spreads until all routers are alerted and thereby trapping the worm within those hosts which are already infected. Once the worm stops spreading, the routers in whose sub-net there is no infection back off from scanning the traffic, turning the white dots back into green, while routers with infected sub-net continue to scan traffic.

The curves for a typical scenario with 8 friends for each host is shown in Fig.2. Initially when the worm starts spreading, none of the hosts are aware of it. But once some one raises an alarm, a large number of the participants are alerted and traffic is monitored. The graph shows that the alert messages spread much faster than the worm and thereby triggering the filter rules at a large number of places before the worm could reach them. We can see that the maximum infection is restricted within 25% of the population. The drop in the infected population is due to the patching of machines against the worm. We can change various parameters such as the number of friends each host has, the spread speed of the worm, the network scanning technique the worm uses, etc., in the simulation to study the effectiveness of our model.

It was also found that if we increase the number of friends, we can restrict the infections to a lower than 25% of the total. But the number of new friends needed is too high to achieve any marked improvement. And in practice, when the number of friends becomes too high, trust becomes a casualty.

So, to further restrict the number of infections, another model, the Hierarchical Model was proposed which is discussed in the rest of this paper.

⁸Term first seen in [18]

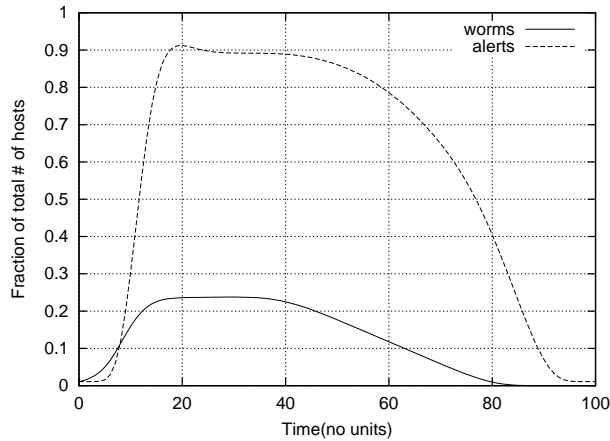


Figure 2: The graph shows how the alert messages catch up with the worm.

5 The Hierarchical Model

This model assumes a tree structure where the internal nodes of the tree are firewalls and leaves are servers vulnerable to worm attacks, Fig.3. The firewalls are assumed to be immune to infections. It is also assumed that we have sensors at the vulnerable hosts that can detect an infection and report it. All nodes at a particular level of the hierarchy have equal number of children. Each level of the hierarchy has a threshold associated with it up to which nodes at that level can tolerate infections amongst its children. Once the number of infection reports amongst its children reaches the threshold, the firewall turns on the filter rules protecting all of its children and in addition, alerts its parent that the sub-tree below it is infected but protected now. This escalation of alerts from one level to the next higher level in the hierarchy and protection of sub-trees takes place successively as the threshold for infections is reached at each node.

False alerts are handled by the firewalls by associating a 'Time to Live'(TTL) with the latest alert that they receive. If the threshold is reached before the TTL expires the above said actions are taken. If the TTL expires before the threshold is hit, all the alerts/infection reports are discarded and the firewalls 'back-off' from protecting its children. This is done for the obvious reason that any action taken to contain a worm involves a cost. And such actions undertaken when there is really no worm is no use.

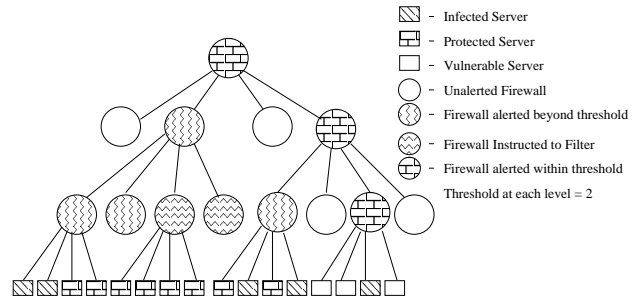


Figure 3: Hierarchical relationship among firewalls and vulnerable hosts.

An extension of this model to handle false-alarms which we haven't yet tested would be the following. The TTL is increased for the next alert for it could be a 'Stealth worm' that is spreading. As we know, Stealth worms spread very slowly. The quantum of increase should neither be too large that we fall prey to false alarms nor should it be too small that 'Stealth worms' propagate. However, there is no definite algorithm to determine this quantum. One could just double the TTL or increase to one and a half times its previous value. It is left to the site administrator.

This TTL could be reset to a desired realistic value once it reaches a maximum limit or after the alerts have been thoroughly examined manually by experts and have been declared to be false alarms.

5.1 Simulation of Hierarchical Model

The Hierarchical Model simulation was implemented in Perl. The simulation was done on a network modeled as a tree with 4 levels. Each level of the tree has 4 children.

The simulation is started by randomly infecting a single leaf node. The rate of infection is fixed at the beginning of the simulation. The exact number of machines that each infected machine tries to infect is determined by using a Poisson distribution, with the mean value (i.e. μ) as the rate of infection. The worm scenarios were simulated with a large TTL value, 1000.

In each time slice, every infected machine tries the infect as many other machines as dictated by the Poisson distribution. Alerts are raised in the same time slice as an infection occurs. And each alert is propa-

gated as high as possible in the hierarchy in the same time slice.

In the case of false alarms, the rate of false alarms is fixed at the beginning of the simulation. Unlike worms where only infected machines can infect others, there is no relationship between one false alarm and the next. The machine that raises a false alarm is determined randomly. All false alarm scenarios were simulated with TTL window sizes varying from 10 to 400 time units.

Simulations were run with thresholds at 75% and 50% of the number of children. That is, if 75% of a node's children have raised alerts, the node takes action.

The structure of network and thresholds were chosen so as to be comprehensible. However, more complex structures with different number of children at each level and different thresholds at each level could also be simulated.

5.2 Discussion of the results

The basic results of two extreme cases where all parameters are identical except the rate of infection which is very high and very low are shown in Fig.4 and Fig.5. These two figures show that the number of infections before complete immunization could take place is almost the same for both the cases.

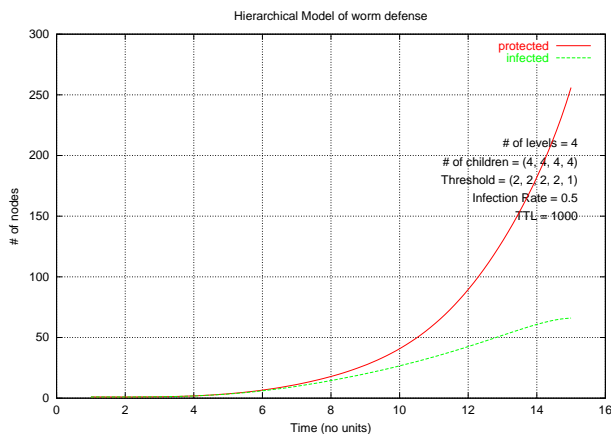


Figure 4: Response for a low rate of infection.

The simulation was done for different rates of infection with 2 different levels of thresholds and the results are shown in Fig.6. As we can see in Fig.6,

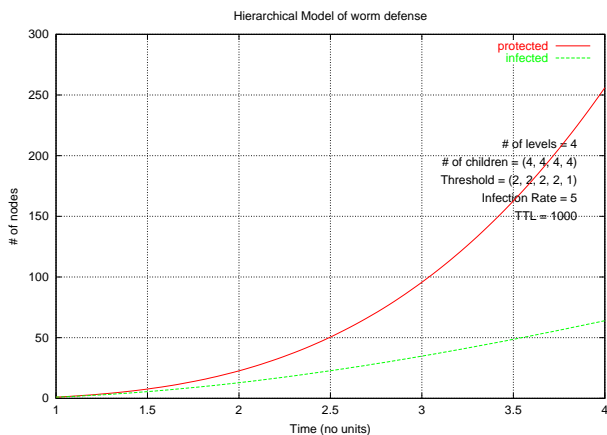


Figure 5: Response for a high rate of infection.

the number of infections before complete immunization takes place is almost the same for varied rates of infection. But the time it takes is different and favorable too, Fig.7. For high rates of infection, maximum possible protection is achieved quicker than for slower rates of spread. This is a direct result of the dependence of the alert propagation on the infection rate. And we also see that the thresholds don't play a part in the time taken as it takes almost the same time to achieve complete immunization for 2 different threshold values. The only thing that varies with threshold is the number of infected machines.

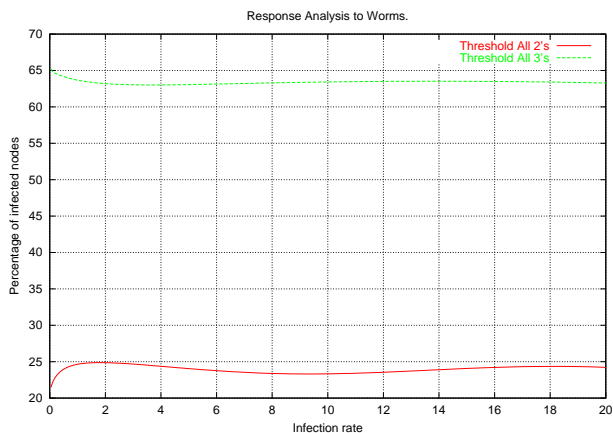


Figure 6: Percentage of machines infected for different rates of infection.

Thus the lessons learnt are :

- For any rate of infection, the number of victims

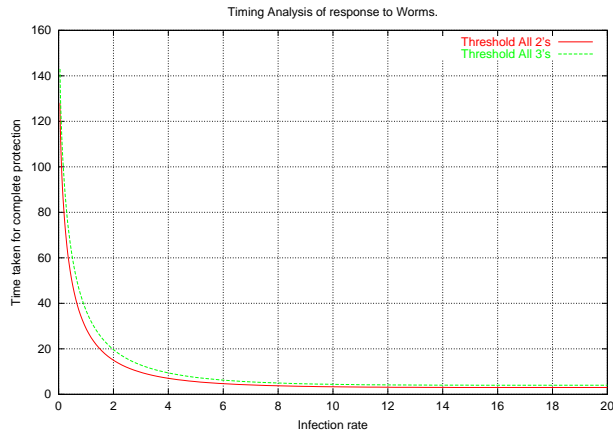


Figure 7: The time taken for complete protection is inversely proportional to the infection rate.

is the same.

- The time taken for complete protection is inversely proportional to the infection rate.
- It is only the threshold levels that makes or breaks the network. A low threshold helps save a lot of machines.

It is obvious that we can't forecast an unknown worm's infection rate. But we can define our tolerance. So, we now have a model which can tell us what should be the threshold, for a given tolerance.

For eg., if we want to save 90% of the Internet in the event of a worm attack, we just need to find out from simulations what should be the thresholds at various levels of the network hierarchy and set the firewall rules accordingly. It would do the job for us slowly for quickly depending on the infection rate of the worm.

5.3 False Alarms

During false alarms the number of machines given protection does not rise steadily as it happens in case of real worms. Rather, the number of machines protected keeps oscillating as the systems backs-off if there are no alerts within the TTL as seen in Fig.8. This oscillation is an indication of a series of false alarms. Or rather it can be considered as an indication of a Stealth Worm in motion which has very

slow rate of spread which is discussed in the next sub-section.

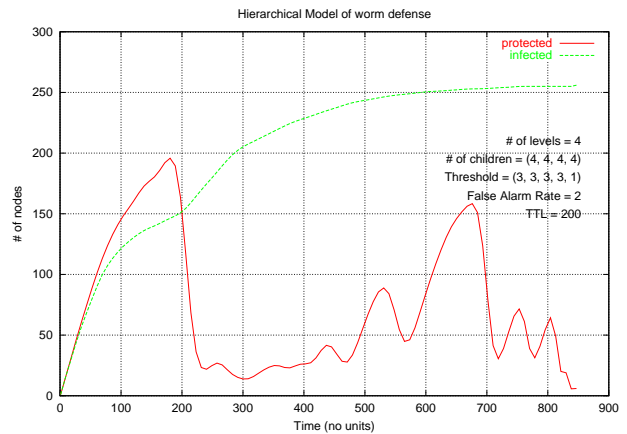


Figure 8: The number of machines that are protected keeps oscillating in case of False alarms.

Fig.9 shows the average number of machines that are given protection in response to false alarms for various TTLs and various false alarm rates. This protection involves a price and can be considered equivalent to a self inflicted DoS. As we can see, if the TTLs are low enough, we have to pay a very less price. But we would not be able to capture Stealth worms as is seen in Fig.11. At the same time, a high TTL would 'cry worm' even for low rates of false alarms and raise costs unnecessarily.

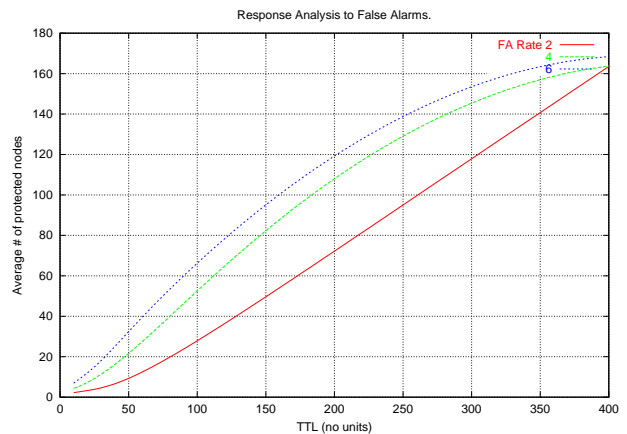


Figure 9: Average number of machine protected for different False Alarm rates for varying TTLs.

5.4 Stealth Worms

We can see the same oscillating pattern in Fig.10 and Fig.11 which were recorded for a Stealth worm simulation with a small TTL of 60.

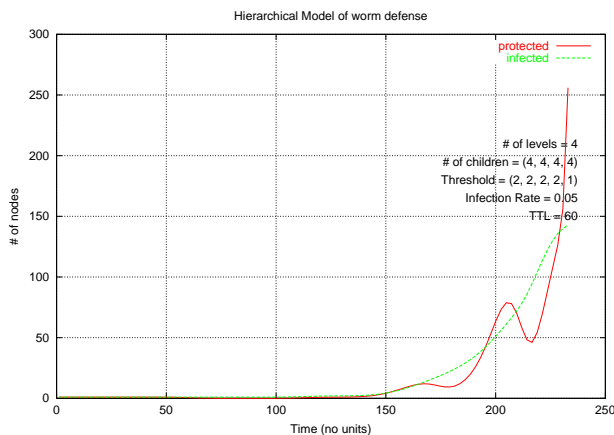


Figure 10: A stealth worm overpowered with a low threshold.

Fig.10 shows a case where the stealth worm is suppressed because of a low threshold of the defense system.

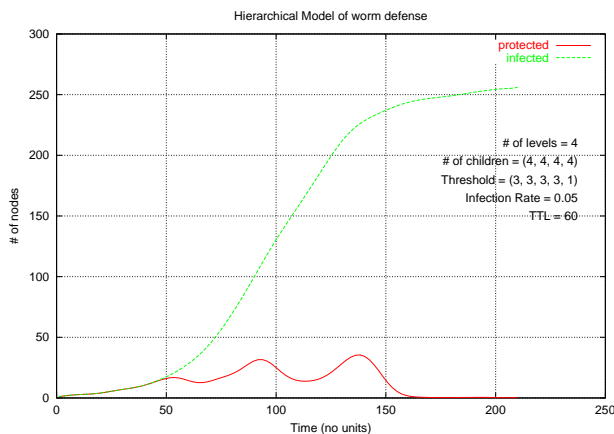


Figure 11: A Stealth worm sneaking in due to a high threshold.

Fig.11 shows a scenario where the protection mechanism is able to sense that something is wrong. But due to the high threshold, the rate of alerts received is just low enough that the TTL expires frequently and our system backs-off reasoning it as a false alarm. This suggests that we need a higher TTL or a lower

threshold, which we addressed above. But a high TTL means a high cost due to false alarms which is unworthy.

Since an oscillating curve means a series of false alarms or a stealth worm which is slow moving, we can afford the luxury of human intervention.

So, we need to arrive at a compromise saying that we will look out for Stealth worms which only move above a certain speed. In any case, slower worms will get exposed as more and more machines are infected, as this will increase the overall rate of infection. So, the TTL in a way, also dictates how many machines we will have to sacrifice before our defense mechanism takes over. We may even lose out all machines before we respond if we choose a very small TTL.

6 Future work

Possible related work includes a study of correlation of data from various sites and develop a model to identify if a malicious activity is really a worm. The required data could be collected from various "intrusion detection systems" deployed at different trusted locations. GrIDS[15] is a good place to start with, for this study.

Developing a prototype to verify these models on isolated networks with real worms will be the next logical first step in the direction of implementing this model in practice.

The load on the network due the alert messages is another important facet of this project that needs to be studied.

Theoretical analysis on the effect of the propagating worm taking up bandwidth to the point where the alert messages cannot get through is in order. This could be done by treating the network as a graph and answering the question, "How many links may get saturated before the alerts cannot reach others?". [4] would form a good starting point for this research.

7 Conclusions

Using this model and the simulations we can determine the thresholds required at various levels of the

hierarchy for a given tolerance of lost machines to handle worms. We can also determine the TTL by looking at the usual rate of false alarms in an environment and how slow a Stealth worm we are interested in stopping. So, with the threshold levels and TTLs thus determined, we can effectively stop worms without losing much due to false alarms.

References

- [1] "<http://www.kryptocrew.de/snakebyte/e/StoppingWorms.txt>". Internet.
- [2] "<http://www.swarm.org>". Internet.
- [3] "<http://www.hackbusters.net/LaBrea/>". Internet, 2001.
- [4] Tuomas Aura, Matt Bishop, and Dean Sniegowski. "Analyzing Single-Server Network Inhibition". In *Proceedings of the 13th Computer Security Foundations Workshop*, pages 108–117, July 2000.
- [5] CERT. "CERT Advisory CA-2001-19 "Code Red" Worm Exploiting Buffer Overflow In Iis Indexing Service DLL". Internet, January 2002. <http://www.cert.org/advisories/CA-2001-19.html>.
- [6] C.G.Senthilkumar. "Worms: How to stop them? - A proposal for Master's thesis.". University of California, Davis. <http://wwwcsif.cs.ucdavis.edu/~cheetanc>, July 2002.
- [7] Mark W. Eichin and Jon A. Rochlis. "With Microscope and Tweezers: An analysis of the Internet Virus of November 1988". In *Proceedings of the symposium on Research in Security and Privacy*, May 1988. Oakland, CA.
- [8] Carey Nachenberg. "Computer Parasitology". Symantec AntiVirus Research Center.
- [9] Carey Nachenberg. "Understanding and Managing Polymorphic Viruses". Symantec AntiVirus Research Center.
- [10] Don Seeley. "A Tour of the Worm". In *Proceedings of 1989 Winter USENIX Conference*, pages 287–304, Berkeley, CA, February 1989. Usenix Association, USENIX.
- [11] John F. Shoch and Jon A. Hupp. "The "Worm" Programs - Early Experience with a Distributed Computation". *Communications of the ACM*, 25(3):172–180, March 1982.
- [12] Eugene H. Spafford. "The Internet Worm Program: An Analysis". Technical Report CSD-TR-823, Purdue University, West Lafayette, IN, USA, December 1988.
- [13] Eugene H. Spafford. "The Internet Worm: Crisis and aftermath". *Communications of the ACM*, 32(6):678–687, June 1989.
- [14] Eugene H. Spafford. "The Internet Worm Incident". Technical Report CSD-TR-933, Purdue University, West Lafayette, IN, USA, September 1991.
- [15] S.Staniford-Chen et al. "GrIDS - A Graph-Based Intrusion Detection System for Large Networks". In *The 19th National Information Systems Security Conference*, volume 2, pages 361–370, October 1996.
- [16] Stuart Staniford, Vern Paxson, and Nicholas Weaver. "How to Own the Internet in Your Spare Time". In *Proceedings of 2002 Summer USENIX Conference*, Berkeley, CA, August 2002. Usenix Association, USENIX.
- [17] Nicholas Weaver. "Future Defenses: Technologies to Stop the Unknown Attack". Internet, February 2002. <http://online.securityfocus.com/infocus/1547>.
- [18] Nicholas Weaver. "Warhol Worms: The Potential for Very Fast Internet Plagues". UC Berkeley, February 2002.
- [19] Tarkan Yetiser. "Polymorphic Viruses. Implementation, Detection and Protection.". VDS Advanced Research Group, January 1993.