Simulating and optimising worm propagation algorithms

Tom Vogt <tom@lemuria.org>

29th September 2003

Abstract

This paper describes a series of simulations run to estimate various worm growth patterns and their corresponding propagation algorithms. It also tests and verifies the impact of various improvements, starting from a trivial simulation of worm propagation and the underlying network infrastructure to more refined models, it attempts to determine the theoretical maximum propagation speed of worms and how it can be achieved. It also estimates the impact a malicious worm could have on the overall infrastructure.

Contents

1 Introduction

Much has been written about worms since Code Red, though they are hardly a new phenomenon. Most of what has been written has been exaggerated and contains little technical details. The technical articles fall in two categories: Coding new or analysing old worms, i.e. implementation details on the one hand. Mathematical modelling, usually through population/epidemic models on the other.

To the best of my knowledge, no in-depth study using actual or simulated worm behaviour has been published so far. This paper closes this gap, though much more work in this area can be done.[1]

1.1 Motivation

The main purpose of this article is to show that we will need new and better countermeasures against worms. I will demonstrate that propagation algorithms capable of ”Flash Worm” speeds are feasible. As such, there is no margin for the current cycle of defending against worms, where we, the security community, wait until we catch a copy, then analyse it, then discuss countermeasures to be deployed manually. In the near future, the game will be over before discussion can even begin, and the worm will have infected the entire Internet before we know what we are dealing with.

1.2 Methodology

All data in this article was derived from simulation. Using simulation instead of mathematical formula allowed me to approximate closer to actual worm propagation, introduce random factors and model adaptive behaviour.

A simple simulation program was written that holds a simplified dataset of a ”scaled down” Internet (one class A network, 16 mio. addresses). Using a network (with shortened, but otherwise proper IP addresses, e.g. ”127.0.1”) allowed me to re-create local preferences such as demonstrated by Nimda. Information about each host is highly abstracted and most real-world issues such as server load, monitoring systems and other artifacts which will intentionally or unintentionally hinder a worms propagation, were ignored. A simple 1% chance for any connection to fail was included as a primitive simulation of general network unreliability.

In the real world, IP address space is not distributed randomly. Large chunks are unused while others are nearly filled. Where you find one server of a specific kind, others are often in the vicinity.

In the simulated network, simple clusters have been added based on class C size net-blocks. For every net-block, a ”preference value” is randomly assigned, using the same model of distribution as described above for individual hosts. If the preference indicates a largely unused address space, there is a 50% chance for each individual address to be automatically unused as well. Otherwise, the normal (random) distribution of hosts applies. If the preference indicates a certain host type, there is a 25% chance that the server will be of this type, otherwise the normal distribution applies. Note that the normal distribution can give the same result that the preference would have given.

Hosts (used IP addresses) appear as abstractions in the simulation. Instead of modelling various operating systems and services, each host is simply considered to be one of the following:

offline which denotes unused address space or systems not currently connected, as well as ”Dark Address Space”[1]. Also, servers behind a firewall that drops (instead of rejecting) connection attempts to the vulnerable port fall for all practical purposes into this category. Connections to these systems will time out, which can take a while. 70% of the address space is unused or offline.

[1] A simulation model for a Code Red II like worm is included in the simulation software SSFNet, release 1.5 - I am not aware of any research on modifying this model with the intend of creating a better worm.

other_service contains systems not offering the service in question, e.g. mail-servers (SMTP) for a worm spreading via a web (HTTP) vulnerability. These systems reply with an RST packet to the connection attempt, allowing the worm to continue quickly to other systems. 50% of the existing hosts (15% of the address space) are assumed to be of this category.

not_vulnerable includes systems running the service in question who are not vulnerable to the virus propagation method, usually an exploit of a specific server software. For example, Apache web-servers were *not_vulnerable* to Code Red. About 47% of the hosts are assumed to belong to this group.

vulnerable these are all systems that the worm can infect, but has not yet infected. Only 1% of the address space (about 3% of the hosts) are considered vulnerable. In light of past virus outbreaks, this number appeared as realistic, if not conservative. Modifications of the fraction of vulnerable hosts will be investigated in section 4.3.2.

infected systems successfully infected by the worm, and now acting as propagation points. Initially, no systems are infected, until at the start of the simulation a given number of systems is chosen as starting points for the worm.

Table 1 shows examples from the simulated network and the distribution of hosts over it. The first network contains a typical large hosting site using a non-vulnerable server software, with a few future victims in between, possibly as test servers. The second network could be another mass-hoster, but one using the vulnerable software, with a couple of immune systems (patched or broken in a way that disables the worm, or using a non-vulnerable version) and some systems with other services. The third is a sparse network with 15% of the addresses used, mostly for a different service. These might be mostly routers or monitoring systems.

Network	Total Hosts	Other Services	Not Vulnerable	Vulnerable
69.1.0/24	160	56	101	3
82.220.0/24	166	40	33	93
103.127.0/24	39	28	9	2

Table 1: Examples for system distribution

Interpretation of these values should not go too far, as it is but a very simple simulation of the clustering effect visible in real networks. It will be interesting later on, though, to analyse whether vulnerable systems in the midst of other vulnerable systems, or as the odd one out in a largely non-vulnerable network have better chances of survival.

1.3 Real-World Comparisons

Code Red, still the most famous of the recent Internet worms, spread to over 350,000 systems in about 13 hours, with the fastest and most visible part of the growth process (from under 10,000 to over 300,000 systems) happening in about 4 hours according to CAIDA[2] or 9 hours if you choose to believe CERT[3].

Code Red II improved on the propagation algorithm with a resulting increase in propagation speed. According to some news media, it spread ”4000 times faster”[4] than the original Code Red. This number is very likely inflated, possibly even arbitrary. Actual analysis is made more difficult due to the continued spread of the first Code Red during CR2s appearance, but hints at a propagation speed only slightly faster than Code Red. However, CR2 spread in a more difficult environment with system administrators being aware of the problem and many systems having been patched.

Sapphire/Slammer improved on the Code Red variants and the later Nimda considerably:

> The Sapphire Worm was the fastest computer worm in history. As it began spreading throughout the Internet, it doubled in size every 8.5 seconds. It infected more than 90 percent of vulnerable hosts within 10 minutes.[10]

This is certainly a performance that a theoretical worm should aspire to reach, or beat.

1.4 Other Reading Material

- ”How to 0wn the Internet in Your Spare Time”[6] is a formal analysis of Code Red, Code Red II and Nimda, and contains some theory on future worms as well. It is heavier on the math than this paper, but limits itself to a simple simulation. It contains many intelligent insights into worm design and propagation algorithms. It also creates the terms ”Warhol worm” and ”Flash worm”.
- ”The Future of Internet Worms”[7] is a broader approach on the subject, including payload proposals and looks at key components of worm capabilities. It gives especially valuable insights into the possible damage a professionally written worm could cause.
- Bruce Ediger’s NWS `http://www.users.qwest.net/~eballen1/nws/` is a network worm simulator written in Perl. It uses a much smaller network (Class B size) but shows very similar results. The website is also much heavier on mathematics and concentrates more heavily on analysis than on possible improvements.

Many other articles have been written about worms in the wake of Code Red and later media events. One good collection is provided by Sans[8].

1.5 Limitations

The simulations does not take into account latency issues, hop-count or connectivity issues. It does make some attempts to consider bandwidth limitations and transfer times in the more refined models.

Since the simulated network is just $\frac{1}{256}$th the size of the Internet, all results must be assumed on that scale, taking into account the growth curve of the worm. For exponential growth, which most propagation algorithms at least approach, that means all times given need to be multiplied by about 2.4, i.e. a worm spreading through the entire simulation network in 2 hours would require just under 5 hours for the real Internet. This *scale factor* should be taken with care, however, as it does not take real-world complications into account at all, and no conclusive proof as to how exactly networks of this size scale up has been presented so far.

2 Theoretical limits

After finding some of the propagation speeds shown below, I also wanted to establish the "speed of light", i.e. the highest possible propagation speed that a perfect worm could reach.

With enough abstraction, this is trivial. Assume that the worm has perfect knowledge and wastes no time whatsoever on scanning unused address space or attacking non-vulnerable systems. Then the only limiting factor to propagation speed is the number of systems the worm can infect in a given time t. Call this propagation rate r. Then the number of systems infected can be expressed as the following series, starting with i systems initially infected:[9]

$n_0 = i$

$n_1 = n_0 + rn_0 = (r+1)^{n_0}$

$n_2 = n_1 + rn_1 = (r+1)n_1 = (r+1)(r+1)n_0 = (r+1)^{2n_0}$

[...]

$n_t = n_{t-1} + rn_{t-1} = (r+1)^{tn_0} = (r+1)^{ti}$

In the special case of $i = 1$(i.e. starting the infection from a single system), solving for t gives the time required to infect a given population:

$t = log_{(r+1)}n_t$

Please note that r is the number of *successful* infections per second. Neither re-infections, nor attempts on non-vulnerable systems are taken into account in this formula.

2.1 Conclusion of theoretical analysis

Looking at some numbers, the perfect worm puts the concept of a Flash worm (10 minutes or less) to shame. At an infection rate of 1 (each system infects one other system per second), a population of one million will be completely infected in under 20 seconds.

Rate	Pop. 100	Pop. 1,000	Pop. 10,000	Pop. 100,000	Pop. 1 mio	Pop. 10 mio
0.2	25.26	37.89	50.52	63.15	75.78	88.40
0.5	11.36	17.04	22.72	28.39	34.07	39.76
1	6.64	9.97	13.29	16.61	19.93	23.25
2	4.19	6.29	8.38	10.48	12.58	14.67
5	2.57	3.86	5.14	6.43	7.71	9.00
10	1.92	2.88	3.84	4.80	5.76	6.72
20	1.51	2.27	3.03	3.78	4.54	5.29

Table 2: Theoretical propagation speeds of a perfect worm

As shown in Table 2, with ten infections per second, a target population of one million would be completely infected in under 6 seconds. Higher infection rates will reduce this further, but for all practical purposes, the difference between 6 seconds and 3 seconds (at 100 infections per second) is negligible, and the return on investment decreases sharply beyond about 10.

While the assumed total knowledge seems entirely theoretical, the Flash Worm concept discussed in [6] is describing one approach to realize this perfection, through pre-scanning of the entire network. Section 4.5 on page 17 will discuss this option.

3 Initial Tests

Some preliminary testing was done on a simplified dataset with a random distribution of hosts. It already shows the typical worm propagation graph from slow start to explosive growth until saturation[2] is reached. It is also interesting to see that the infection rate tops almost precisely when a 50% saturation is reached. The other graphs will support this conclusion, and it seems apparent to the author that it can be mathematically proven. This carries consequences to the development of detection and alarm systems, as does the fact that once the growth is visible on the chart, the explosive growth phase has almost begun already.

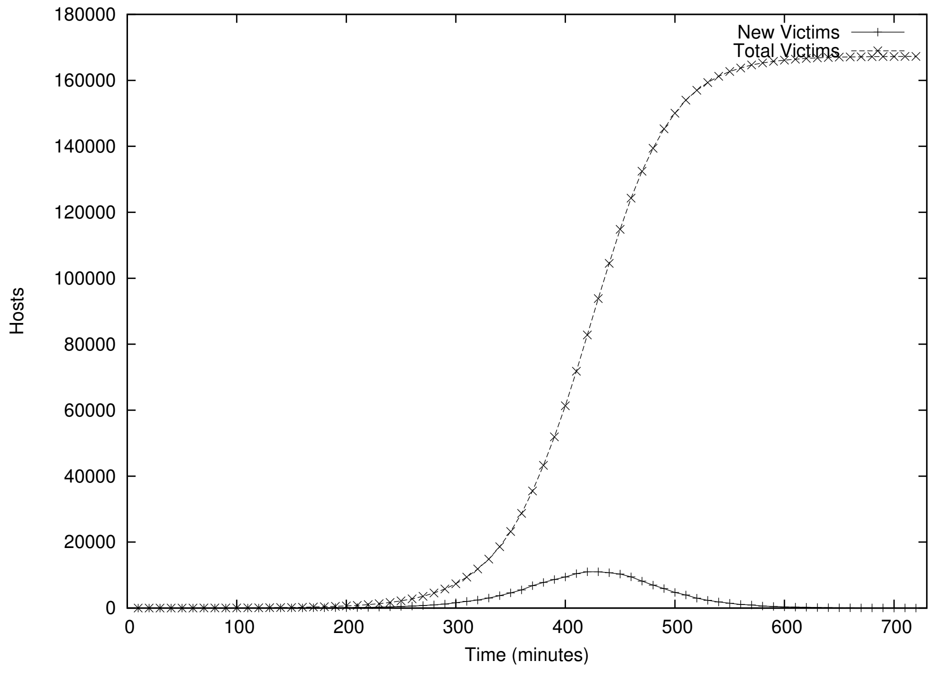

Figure 1: Initial, random dataset with random propagation algorithm

This dataset is based on the following data:

- The distribution method discussed resulted in just over 160,000 vulnerable systems in a network of about 5 million active hosts. At at about 3%, they certainly are a small minority.
- Timeouts for unreachable (offline/unused) systems is 30 seconds, 1 second is used up for probes to non-web-server systems and 2 seconds are needed to infect a vulnerable system. This includes all overhead. The worm uses one thread, no parallel processing.
- A snapshot was taken and converted to a data point for the graph every 10 minutes.
- 5 systems were initially infected at time 0.

The interesting part of the worms growth, that which I call the "explosive growth phase", takes about 4 hours in this graph, just like Code Red did. However, the simulated network is a reduced-size network, and applying the scale factor mentioned in section 1.5 on page 5 results in a time frame of 9 to 10 hours even under ideal circumstances.

[2]Saturation shall be defined as over 90% of the infect-able systems actually having been infected.

3.1 Variations in initial infected systems

Changing the number of initially infected systems shifts the curve, as expected. Various discussions on advanced worm design mentioned pre-infecting a large number of systems as an important step to speeding up worm propagation. However, the data clearly shows that unless a significant part of the vulnerable systems is pre-infected, there is not much difference to the actual propagation. Since pre-infecting considerable amounts of systems carries a high risk of discovery, it appears to not be a winning strategy.

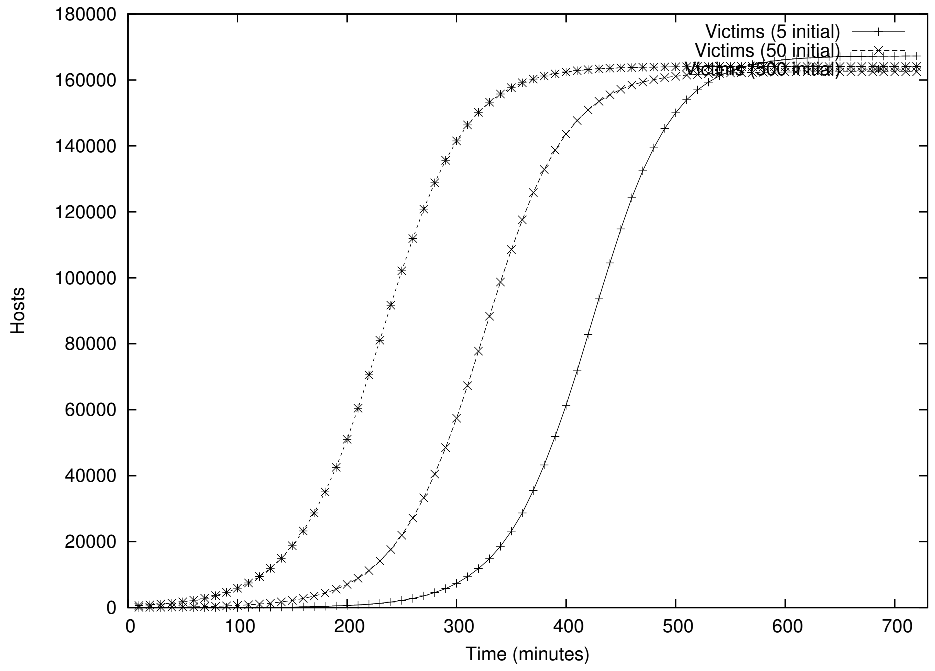

Figure 2: Random dataset and propagation with various numbers of pre-infected systems

While the 500-preinfected-systems graph starts visibly much earlier than the 50 and the default (5 systems) graphs, there is very little difference in the explosive growth phase, and the total time gained is just over 3 hours[3]. Infecting 500 systems manually, and adding a trigger or other mechanism to have them start the worm simultaneously, will take much longer than just starting with a few systems and waiting the same time that would have been spent on coding, not to mention that the resulting worm will be smaller and simpler, thus less prone to have bugs.

We can conclude that, at least with random propagation, any effort invested in pre-infecting systems is a waste of time. A worm will spread just as well and almost as quickly from a single or few systems. In practical terms, the chances of detection are fairly small during the "slow start" period, because of the small number of systems infected and the side-effects of propagation being too low to trigger any alarms.

3.2 Reducing timeouts

Since it was assumed that 70% of the address space is either unused or offline (or firewalled with a drop rule, which will result in the same behaviour as an offline system), connection attempts to these systems will go unanswered and the socket will wait for a timeout before another system can be scanned, at least if the worm is using TCP connections[4]. These timeouts are the longest

[3]The curves do not match near the end because each simulation run randomly generates the network anew, which results in small differences in the total number of infectable systems.

[4]Sapphire/Slammer used UDP, the main reason for its speed

time factor, since a timeout takes much longer than transmitting even a comparably large worm code over even a slow connection. It appears obvious that reducing the timeout would result in a considerable increase in speed for the worm.

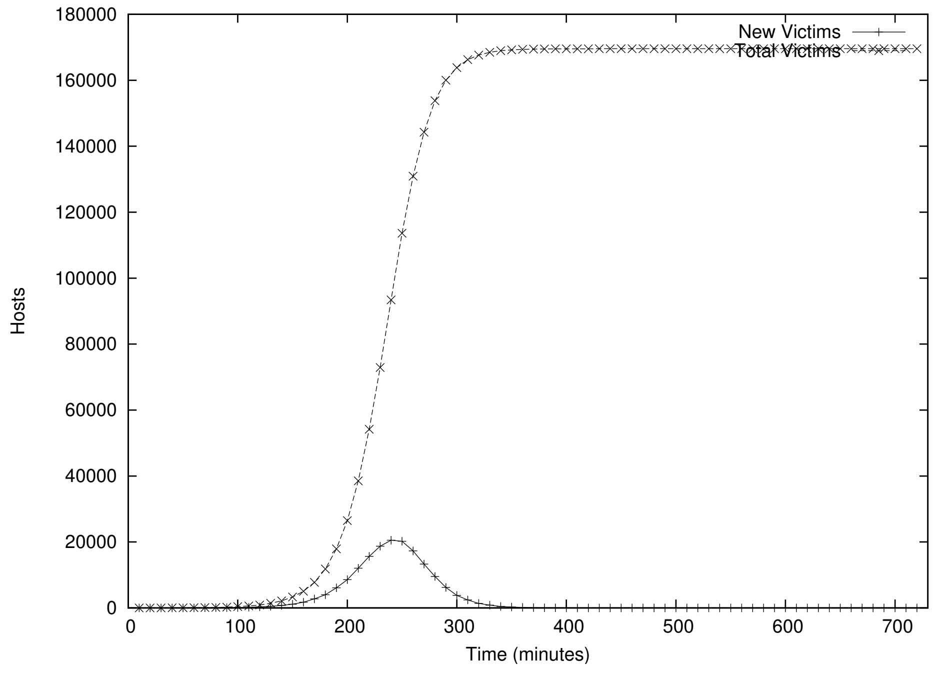

Figure 3: Random dataset and propagation with reduced timeouts

Reducing the timeout from 30 to 15 seconds (while still using 5 pre-infected systems as in the very first study) results in an accelerated propagation as shown in Figure 3. Not only does the noticeable growth part of the graph start much earlier, it is also steeper, or as can be seen from the new victims line, the number of systems infected per cycle is considerably higher. This way, the explosive growth cycle lasts less than two hours. Even ignoring time-zones, this would likely overwhelm the reactive capabilities of the administrator- and anti-virus community.

Various technologies exist to reduce the timeouts considerably. The worm could bring its own low-level TCP/IP processing, or it could simply change the operating systems settings.

A large number of arguments can be made in favour of reducing timeouts considerably, even below the 15 second value used for the simulation above. Systems slow in answering are either a large number of hops away, on a very high latency pipe, or overloaded. The last two do not make good virus propagators anyway, while the first can be ignored as it will very likely be infected by a system closer by later on.

3.3 Parallelised probes aka multi-threading

Probing the address space is intuitively as well as shown by the previous graphs the most time-consuming part of the propagation. Optimisations of probing will therefore go a long way to speeding up worm propagation.

Scanning only one potential victim, possibly waiting for a lengthy timeout is sub-optimal given that all worm host platforms are easily capable of opening many connections at once. In fact, all recent worms have been making use of parallel threads.

Figure 4 shows the propagation graph using the default parameters, except that each client/victim runs 10 threads in parallel. The threads are not optimised to avoid collisions and may rescan addresses already scanned or infected by different threads. However, due to the size of the address space in comparison to the amount of scanning done by a single client (at most 60 addresses per minute and thread, or 6000 addresses per snapshot, out of 16 million) this should not have a noticeable impact on the results.

Please note that this graph has a resolution of 2 minutes and shows only 90 minutes total, instead of 12 hours with a 10 minute resolution. On the 12-hour/10-minutes scale used until now, there would have been very little to see except an almost vertical line.

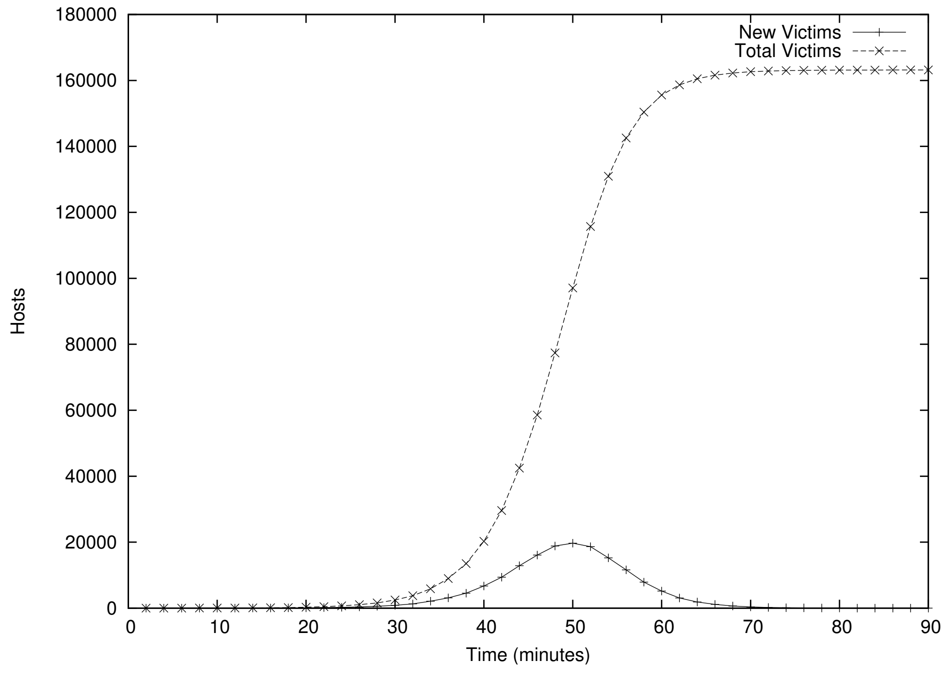

Figure 4: Random dataset and propagation with multi-threading

The result is obvious, with saturation reached in one hour, and the explosive growth phase taking but 30 minutes. This would translate to less than two hours on an Internet-sized network according to the scaling factor discussed on page 5 - faster than the actual speed of all but the Slammer worm, even though it uses only 10 parallel threads. This does not answer the question of why the real-life examples we know of were so inefficient, even though they were using parallel processing at much higher levels (100 threads in Code Red, up to 300 for Code Red II[11]). Section 4 on page 12 will discuss further why this is so and how a more realistic simulation will be realized.

Even though existing worms already employ considerable multi-threading, recent research[12] allows parallelism levels unheard of before. Scanrand claims to be able to map a class B network within seconds[13], with minimal resource waste. Parallel operations allow the worm to continue spreading even while waiting for replies or timeouts. In fact, both can be very easily combined into a single concept, which aims to minimise the worms idle periods.

3.4 Combining threading and reduced timeouts

The two most effective algorithm changes identified so far, reduced timeouts and multi-threading, can easily be combined. The result is shown in Figure 5.

The worm using both multi-threading and reduced timeout reaches saturation in just over 30 minutes in this simplified simulation. At the peak of its propagation, it infects over 30,000 systems per minute. The amount of scanning alone would probably bring down entire networks if it happened in the real world.

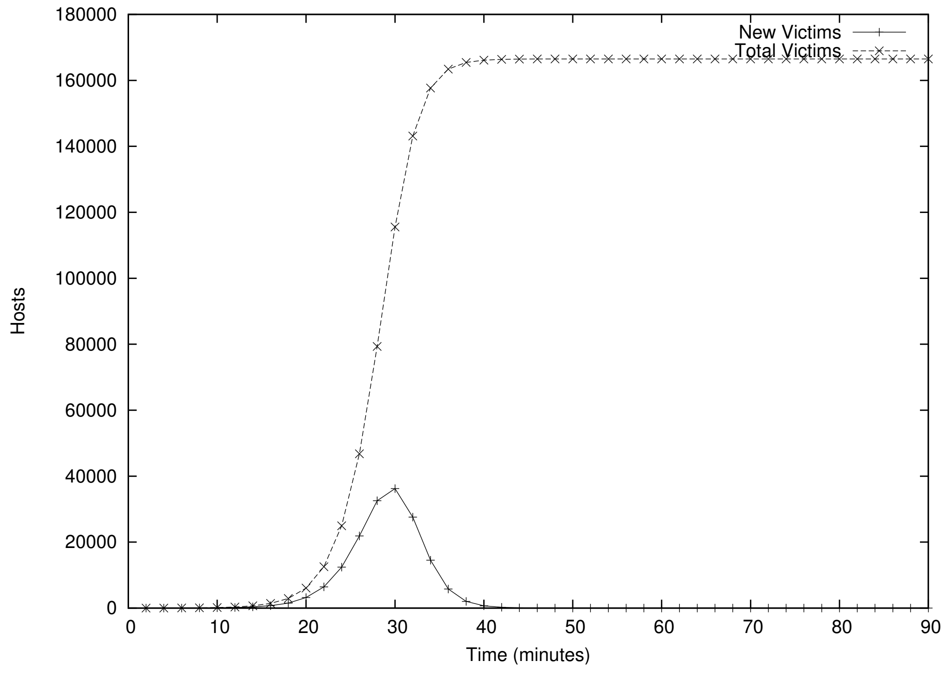

Figure 5: Multi-threading, parallel worm with random propagation

While this figure shows a considerable improvement in propagation speed already, it does nothing to use the synergy possible with changes in the basic approach. The worm is still using the most basic algorithm to spread, namely targeting random systems and sending a copy of itself in their direction.

Also, this simple simulation did not yet take into account issues such as bandwidth limitations, which are likely to dramatically reduce the worm propagation speed. However each system in this simulation is sending out a mere 10 probes per second, which is unlikely to be a major factor even for slow connections. Not the individual connections, but the bandwidth available at uplinks and on the backbone will be the bottleneck.

Other system resources are likewise not likely to be limiting factors. Using 15 seconds timeout, each infected system will at most have 150 connections open at any time, plus whatever it regularly has. Actual propagation of the worm will be a more important factor. At the assumed 2-second transfer time, the worm will require a bandwidth of up to 5 times its own size per second (i.e. a 200 KB worm needs up to 100 KB/s). This value will seldom be reached, however, as most of the time of an individual system is spent with scanning, not sending out copies.

4 Improved propagation algorithms

All of the simulations so far have used a random propagation algorithm, where the worm selects its next target at random. The perfect worm described on page 6 as well as the theoretical Warhol worm of [6] assume a more intelligent approach, however. Several of these can be simulated.

It should be obvious that an improved propagation algorithm will speed up the worm. However, the exact extend of various changes has not been the focus of any research so far.

4.1 Local preference

Code Red II as well as later worms have shown a simple improvement of the random propagation algorithm, based on the fact of clustering. This was described as part of the simulated network on page 3. The idea is that real world computing centers are often mono-cultures, or at least show a strong preference for one type of system. Given the administrative cost associated with running a variety of systems instead of one type, this does make sense from a business point of view, and it also allows a worm to propagate more easily by looking into its neighbourhood first.

Running the simulation with an algorithm that shows a preference modelled closely after the Code Red II behaviour results in the graph shown in figure 6.

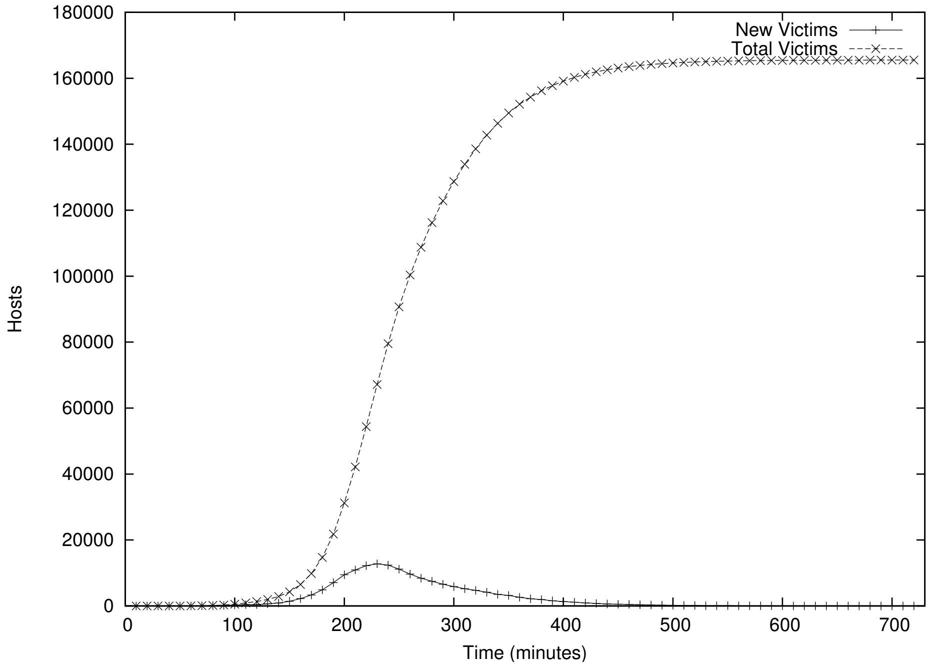

Figure 6: Local preference propagation

This simulation was run with the same values as the initial test (figure 1 on page 7), except for the propagation algorithm. Note that the worm with a local preference algorithm begins his assault much sooner, after about 200 minutes instead of the 300 minutes that the randomly spreading worm requires to get going. It also propagates slightly faster during the explosive growth phase, especially prior to the peak.

Also of interest is the fact that the graph shows a steep left, and propagates only slowly to the last few percent of the victim population. The algorithm thus gains the worm an initial increase in propagation speed, at the price of a slowdown later.

In another simulation the most efficient improvements from the preliminary tests were combined with the local preference algorithm. Using the same values as for figure 5, except for the propagation algorithm, results in the graph shown in figure 7.

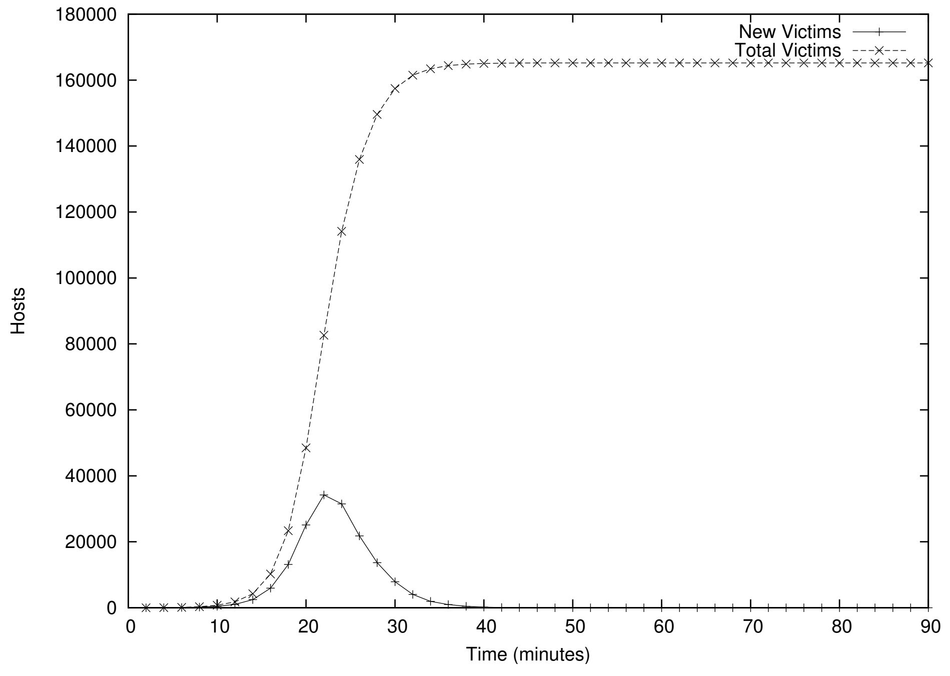

Figure 7: Local preference with multi-threading and short timeouts

The result is in part as expected, namely with an earlier start of the explosive growth phase at about 15 min instead of 22 in figure 5. However, the worm is only slightly more effective, reaching saturation only minutes earlier than its competitor.

From a visual analysis of the graph it appears as if the local preference algorithm has been dominated by the timeout and threading changes, though it still does succeed in shifting the graph considerably to the left. Further analysis was not done on this phenomenon.

4.1.1 Considerations on local preferences

In a real-world situation it is likely that a local preference algorithm will show a stronger improvement as compared to a broader approach, as networks in the same net-block are often geographically and network-topologically close, resulting in faster and more reliable communication, which in turn will speed up both the probing and the infection itself, especially once load on the backbone due to worm activity is reaching critical levels.

4.2 Sequential Scanning

Used by the Blaster worm of August 2003, this algorithm lets each newly infected system choose a random Class C network to start from (40% chance of selecting its current one), then scans sequentially from there. Initial assumptions conclude that this algorithm should be worse than random scanning, as they will create a lot of redundancies. However, like local preference, this algorithm has the advantage once it found a network densely populated with vulnerable systems. Here is the initial graph, all values at their defaults, except for the resolution which has been halved because otherwise the worm would be barely visible:

As can be seen, the simulation confirms the assumption that this algorithm is considerably worse than a simple random target selection would have been. Simulation runs without the local

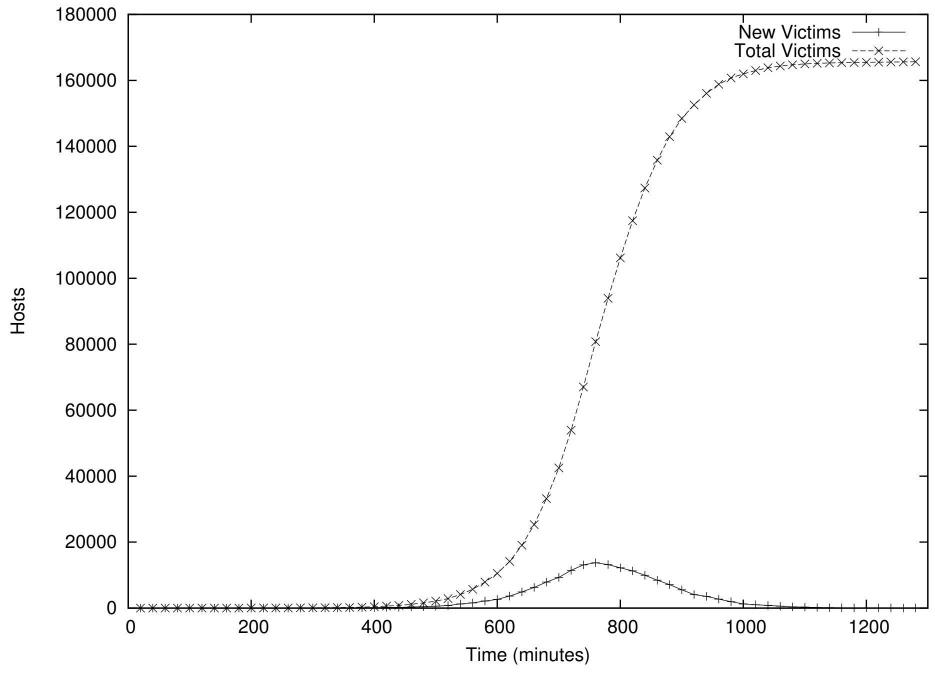

Figure 8: Sequential-scanning worm

preference part also suggest that this addition actually hurts the propagation speed, and is the main factor for the slowdown.

Repeated simulations runs have shown, however, that this algorithm is very susceptible to chance. If it manages to hit one or more networks with many vulnerable hosts early on, it spreads considerably faster than during other simulation runs where it has no such luck. In fact, one very lucky run was almost as fast as the random target selection algorithm. The graph above was chosen as an average example, erring on the side of caution.

The actual data from Blaster's propagation confirm these results.

4.2.1 Improved Target Selection

All the simulations so far assumed that the worm fires itself completely at both vulnerable and non-vulnerable systems. Blaster did not do that. If it failed to infect a system, the TFTP transfer would not happen. Thus, infecting a vulnerable system would take less time then trying to infect a non-vulnerable system. A series of simulation runs were conducted with a parameter choice that was selected in order to approximate Blaster. It is slower on infecting vulnerable systems (3 sec. instead of 2 sec.) but faster when it hits a non-vulnerable system (1 sec. instead of 2 sec.).

The graph of this experiment is not included as it is almost identical to the previous graph. There is a barely noticeable improvement in propagation speed, but it pales in comparison to other changes, and it does not compensate the decrease the sequential-scanning algorithm introduces.

4.3 Better Vulnerabilities

It has been mentioned[14] that some actual worms were hindered by their own bugs or choice of vulnerability to exploit. Even though the infection vector is often not a matter of choice for the worm author, it should not be left out of this treatment.

4.3.1 Higher infection speed

The main strength of the Sapphire/Slammer worm was its infection vector, using a single UDP packet. In addition to not having to go through a TCP handshake and not having to keep a state, sessions and sockets allocated, this also allowed for a faster infection of vulnerable targets than the several KB-sized worms of the Code Red variant could manage, especially with hosts already under heavy load or with bad network connections.

It was not to be expected that this would have a severe impact on propagation speed, in light of the fact that timeouts have already been shown to be the major factor[5]. In fact, a simulation run with infection time requirements halved resulted in only a marginal increase in speed, with the peak shifting forward about 20 minutes, from 440 to 420. Infection speeds were thus not evaluated in detail.

4.3.2 Larger vulnerable population

Obviously, using a better exploit as the infection vector would lead to a larger infectable population first. Ideally, a worm would attempt to be able to infect as large a part of the host population as possible.

In a simulation, shifting the distribution of hosts accomplishes this effect, by declaring non-vulnerable hosts or such with a different service (to take into account worms with multiple infection vectors) as vulnerable. Figure 9 shows the resulting graph from the simulation with four times as many vulnerable hosts as the model used elsewhere in this paper, twice the resolution (one tick being 300 seconds) and otherwise the same parameters as in figure 1 on page 7.

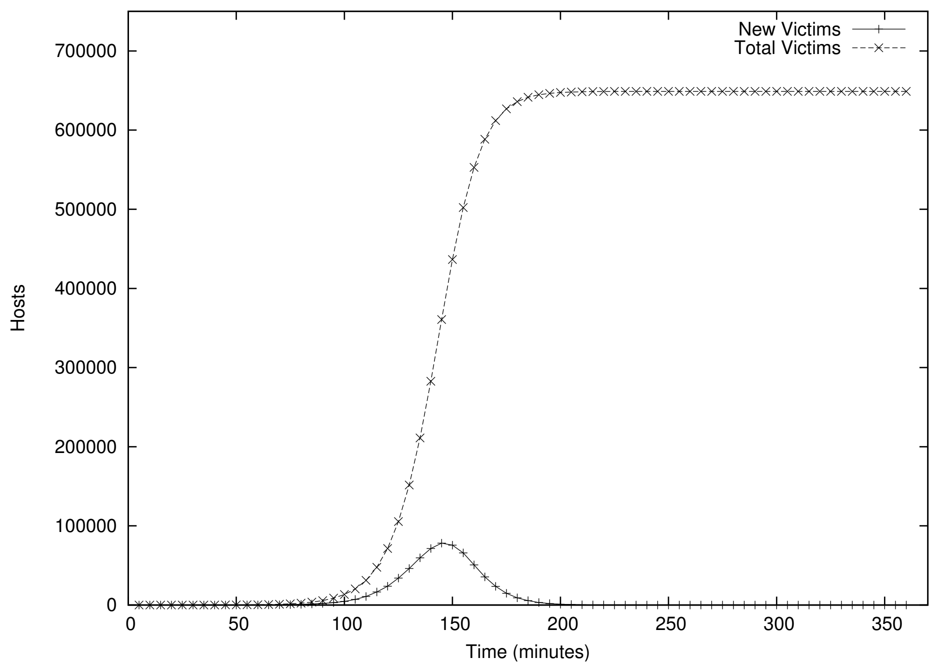

Figure 9: Worm with a higher vulnerable population

Unsurprisingly, a larger vulnerable population leads to a considerably faster propagation, as more hosts are found more easily. This fact is well known in medicine when it comes to epidemics and is why vaccination is often government-enforced.

[5]This also explains Sapphire/Blasters speed: As a UDP worm, it could work on a "fire and forget" principle, eliminating timeouts completely.

It can, however, also be seen that a better vulnerability alone does not make a Flash worm. Quadruplicating the vulnerable population does not speed up the propagation of the worm by an equal factor, but only by a factor of about 2.5.

4.4 Subdividing

Much has been written about subdividing search space, where each worm spawns children of itself that search only a part of the address space. Using this method, the worm as a whole searches the address space much like a binary tree search, which is known to be more efficient then a linear search.

A first implementation was made that subdivides locally, i.e. each child will inherit the net-mask from its parent +1, centered on itself. The initial worms will scan the entire network, the 2nd generation only half, the 3rd only a quarter, etc. When the net-mask reaches /20, it is reset to /0.

This algorithm surprisingly offers no speedup, and is comparable to a pure random propagation. It does exhibit a strong dependency on random factors and a less well-defined growth curve compared to other algorithms.

The graph in figure 10 was created by a modified algorithm that increases the mask by 4 points per tick instead of just one.

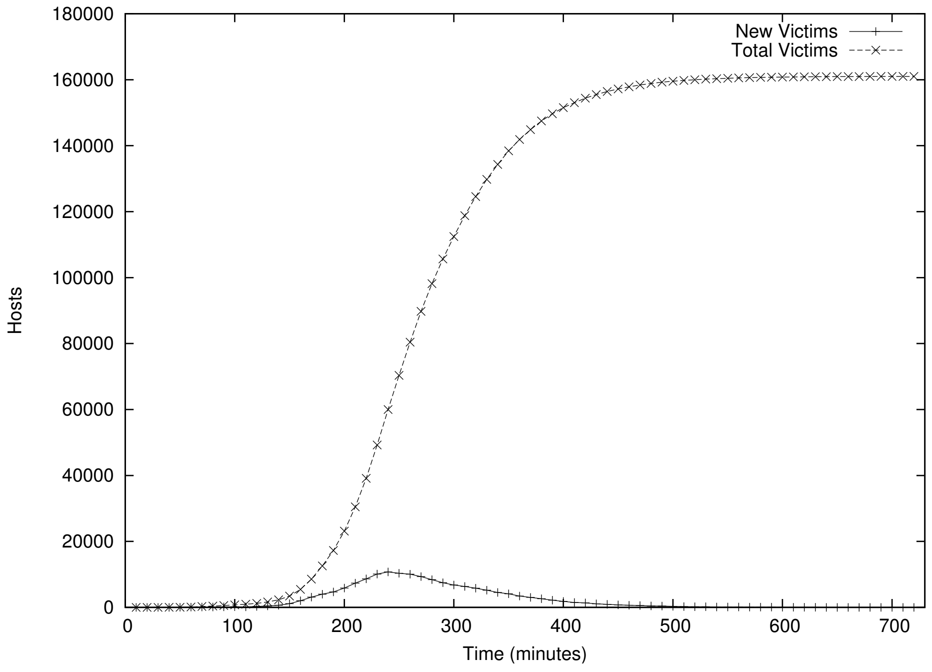

Figure 10: Subdividing the target space

This does result in a left-steep curve and a considerably faster propagation, much like local preference does. In fact, subdividing can be seen as a special case of local preference, where the preference stays constant, but is different for each instance of the worm, depending on generation count.

4 seems to be near the optimum number here, as experiments with both lower and higher increases did not improve the propagation speed any further.

4.5 Pre-scanning

With the advance of stateless scanning algorithms as utilised by scanrand and mentioned above on page 10, it has become possible for a worm to avoid TCP timeouts entirely, taking away one of the largest slowdown factors.[6]. The following simulation is using a theoretical worm that uses a stateless scanning algorithm and proceeds to infection attempts only when it receives a reply. It thus has no timeout whatsoever and is limited only by the number of probes it can send out and the number of replies it can handle.

The algorithm used in the simulation works as follows: The worm picks a /26 network (64 addresses) instead of a single target. It probes this network, which is assumed to take 10 seconds, a value taken from short experiments with scanrand and believed by the author to be good enough to reach a reliability as follows or better: Systems running the service in question (immune, infectable or infected) have a 95% chance to be detected. Systems up, but running a different service have a 1% chance to show up as false positives (wasting one second for the worm). Unused or offline systems will never show up.[7]

Next, all the detected systems are infected, at the same speed as before (2 seconds per system). Again, the first simulation of this new algorithm uses no multi-threading. Nevertheless, it is so efficient, that the graph resolution has been increased to 30 minutes at 30 seconds per tick.

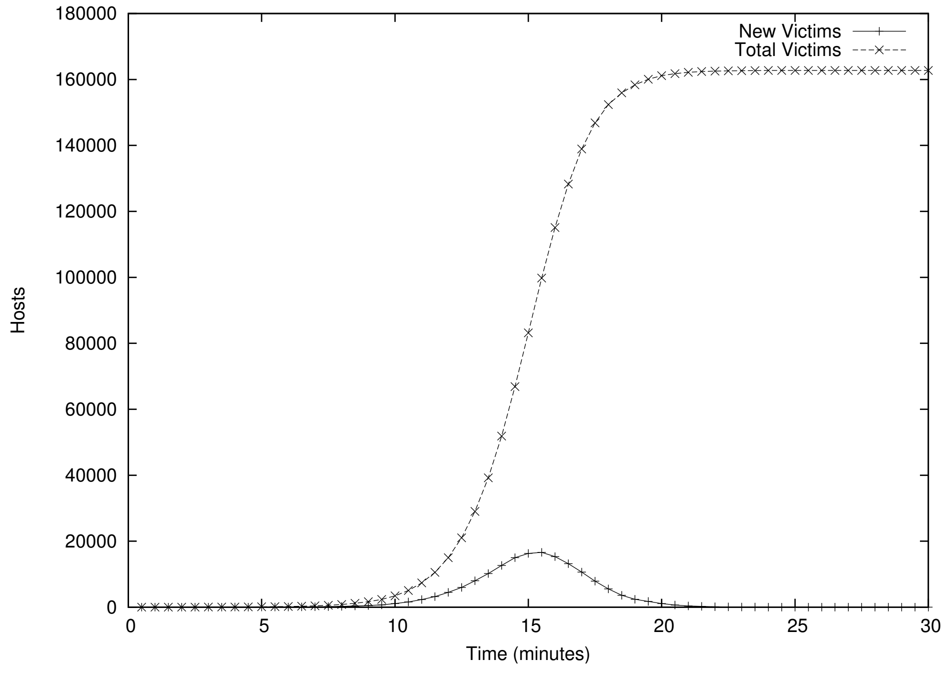

Figure 11: Pre-scanning /26 networks

The main conceptual difference of this worm is that entire networks are attacked instead of individual hosts. In the simulation, /26 network were targeted, and the worm reached saturation in under 20 minutes. No experiments with larger net-blocks were conducted.

Scanning, especially if it can be done efficient, has thus been shown able to improve a worm's propagation speed considerably, by eliminating the timeout delays which have been shown as a major factor already. One other factor is not shown in graphs so far, but deserves mentioning:

[6]If 70% of the address space is unused, and a timeout is 30 sec. compared to at most 2 sec. for an infection or failure, attempting to connect to non-existing hosts takes about 97% of the time.

[7]A simulation accounting for LaBrea or other tar pits would have been interesting, but outside the scope of this experiment.

The number of probes sent out in total and the amount of bandwidth consumed. Section 7 will review this data.

4.5.1 Revisiting initial infections

Even though section 3.1 showed that pre-infecting a number of systems has but a minor effect on the worm propagation speed, the existence of effective scanning algorithms offers new approaches. Instead of manually infecting dozens or hundreds of hosts, the worm could spread from a few hosts, but target networks with many known-vulnerable hosts initially.

The simulation of figure 12 employs such an initial round of pre-infection. For each initially infected system (10 were used), one round of attacking a randomly chosen network with at least 50 vulnerable hosts was initiated at time 0. This starts the worm out with at least 500, more likely 1000 or more infected systems. As seen in section 3.1, the effect of increasing the initially infected systems consists of accelerating the initial phase of the worm, where it otherwise shows no visible activity.

As the graph shows, the simulation is finally approaching the speed predicted in the Flash worm concept, reaching saturation in about 10 minutes.

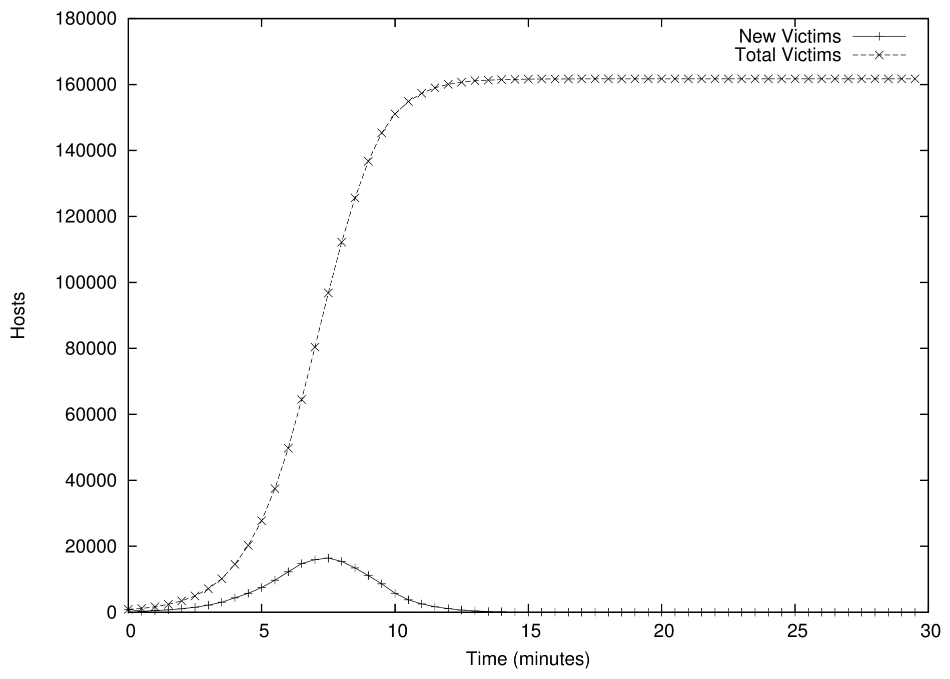

Figure 12: Pre-scanning with pre-infection

4.6 Adaptive behaviour

There are many theoretical forms of adaptive behaviour, but in this article only methods to improve propagation speed are of interest. Of that subset, only those that the worm can conduct autonomously are worth examining, as the purpose is still to develop an algorithm that is too quick for human interaction.

4.6.1 Avoiding re-infections

So far, the worm has not changed its behaviour depending on what it finds. Ideally, though, it would skip already infected systems. If the propagation algorithm aims at entire networks, it should even skip net-blocks. This does, of course, open up the way for an easy antidote. Depending on what the purpose of the worm is, and whether or not it can be completed before an antidote is likely to arrive, this does not have to be cause for much concern, though.

The following worm will use pre-scanning as in section 4.5, except that it is assumed that infected systems can modify their hosts in such a way that the scan will identify them. When a worm finds an already infected system in the net-block it scans, it will skip the net-block and scan another random target with a 90% chance. A 10% chance to attack the already infected block anyways was left in order to get systems infected that slipped through the earlier attack.

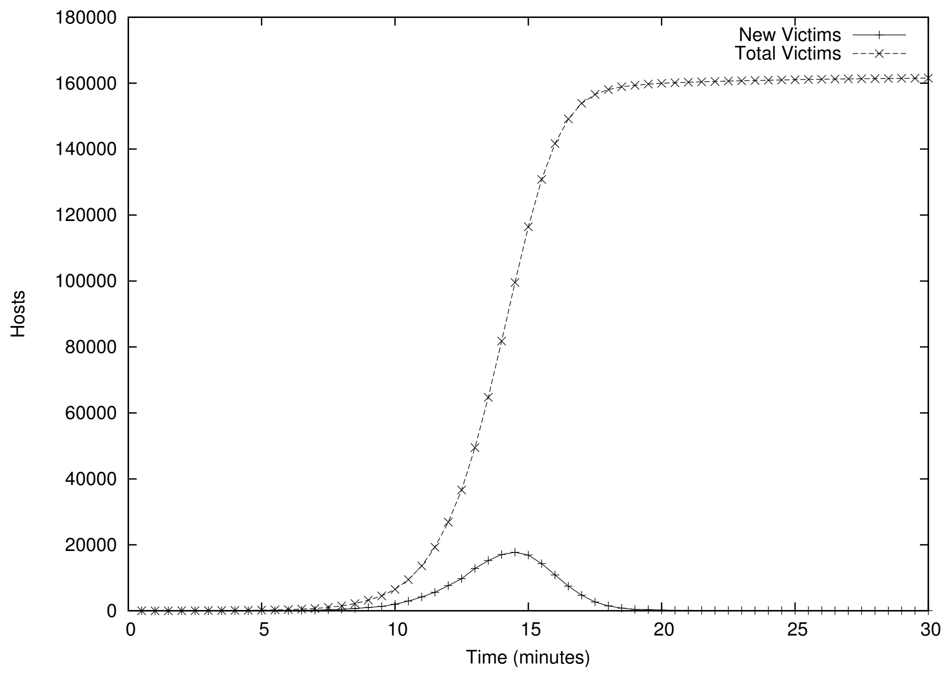

Figure 13: Adaptive behaviour of a scanning worm

This added intelligence makes the propagation curve steeper, resulting in a speed gain of about 10%.

4.7 Do-not-enter Zones

It could be demonstrated that a worm with more information is faster. One piece of information is readily available and would almost certainly help the worm speed up considerably, as it has also been demonstrated that offline systems are a major time sink: If the worm could carry a list of unassigned or unused networks, i.e. know where there simply are no targets to be had, it could stop wasting time there and concentrate on the populated parts of the network instead. Of course, this means more data for the worm to carry, which makes it larger. However, a reasonably short list of large net-blocks (class A or B) would do and would not be very large.

Due to the simplicity of the cluster algorithm used in the simulation, a simplification of this process had to be used, as there are no entirely unused net-blocks in the simulated network. For the following graph, it was assumed that the worm knows about *all* networks that have zero vulnerable hosts and less than 30 hosts total. This happen to be about 1.5% of the address space, or the equivalent of 4 class B networks, certainly an amount of information that even a small worm can easily carry.

Even though a speedup was to be expected, the actual improvement is quite surprising given that the worm possesses only a few bytes of information. More information can be expected to

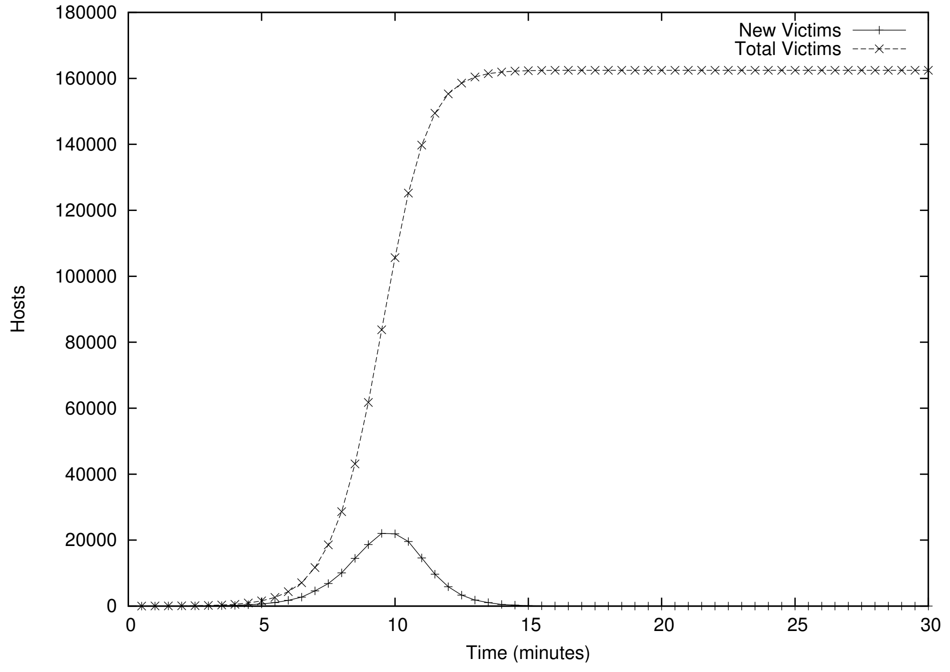

Figure 14: Pre-scanning worm with do-not-enter zones

result in even more improvements, but it stands to reason that the large unused net-blocks are included in this figure and adding more data would increase the worm disproportionately. In addition, a short simulation with twice as many do-not-enter networks showed only a marginal performance increase.

5 Advanced Worms

The following worms are based on the results from the experiments above, but were simulated on an improved version of the simulation software. This allowed for greater precision and a maximum resolution of one millisecond instead of one full second.

There were also technical reasons that required a partly rewritten software, as the simpler version would have required excessive amounts of memory.

5.1 The Advanced Worm

Combining the most effective methods found so far, a good worm should:

- Use a fast, stateless scanning mechanism to find its targets
- Start with known-vulnerable net-blocks
- Avoid known-empty and already infected net-blocks
- Run multiple parallel threads
- Be small and infect quickly

The worm should, of course, also avoid those methods shown to be ineffective, and it should not depend on unknown variables or pure chance. Some of the algorithms visited above have a small chance of being much more effective then the averaged graphs demonstrate, but they also tend to be much slower when they are out of luck.

It is easy to simulate a worm following the above conditions, and still reaching a frightening average propagation speed. The first simulation result, which uses *no* multi-threading, is shown in figure 15.

This worm starts with 10 initially infected hosts, adds a single round of spreading into known-vulnerable networks, then propagates using the pre-scanning algorithm with both do-not-enter zones and skipping net-blocks if it finds that they are already infected. It takes 500 milliseconds to infect a host, and 200 to detect (and ignore) a non-vulnerable one.

Please note that the timescale is in seconds, not minutes.

A local preference part was tested, as it was assumed that it would speed up the early propagation phase, but simulation runs revealed that while it did that, it also slowed the entire propagation down, in result delaying saturation by about 2 minutes.

The slow start of this worm - the curve is slightly steep at the right - could not yet be sufficiently explained, except by the theory that its initial start is artificially inflated due to the pre-infection round.

Even taking the scaling factor into account and erring amply on the side of caution, a worm like this should be able to infect the current Internet in less then half an hour and would take less than 10 minutes to go from first attention to critical mass. It is very unlikely that the security community could stop or even delay it. Aside from some exceptionally paranoid network administrators who pull the plug at the first sign of trouble, the worm would be able to deliver its payload with near-certainty.

A few simulation runs were conducted with more initially infected systems, to eliminate the early slow start, but there are only a few percentage points of speedup in two, five or even three times the numbers.

Network load and hard to simulate real life complications will have a larger impact on propagation speed at this point then further refinements of the algorithm.

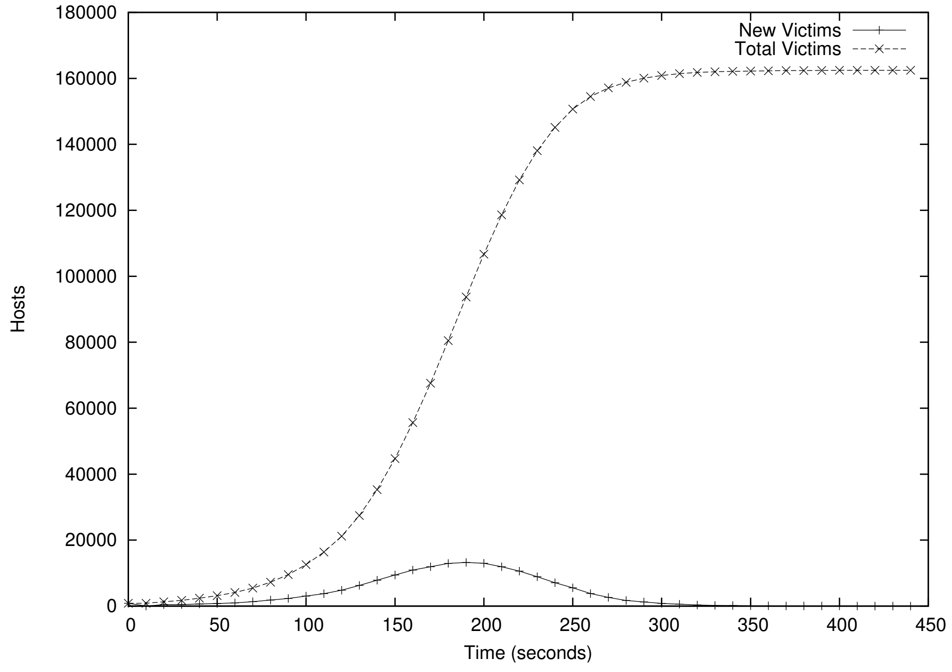

Figure 15: Advanced Worm

5.1.1 Multi-Threading the Advanced Worm

The worm in figure 15 has one shortcoming that has already been found to be a slowdown factor: It is single-threaded. The speed gain from the initial worm (figure 1 on page 7) to the multi-threading worm (figure 4 on page 10) was more than a factor of five, for ten threads. If a similar gain can be made with a more advanced worm, the result should be able to beat the Flash Worms.

Indeed, it does, as figure 16 shows, where it reaches saturation in just over one minute. Even with careful extrapolation, it would do the same on an Internet-sized network in less than five minutes.

The performance improvement is smaller than for the initial worm, only about factor 3. The return-on-investment is diminishing even stronger for additional threads. A simulation with 50 threads brought a gain of roughly factor 1.5, moving the saturation point to just under 60 seconds. No further experiments with multiple threads were conducted.

Comparing this worm to the ”speed of light” from page 6 shows that the advanced multi-threading worm is coming close. At its peak it infects 34,000 hosts during a 5-second interval, from 17,500 then infected hosts. This computes to a rate r of of about 0.389. Returning to the formula this solves to:

$t = log_{(r+1)} n_t = log_{1.389} 162000 = 36.506$

Giving 36.5 seconds as the time until complete infection of a perfect worm with the same infection rate.

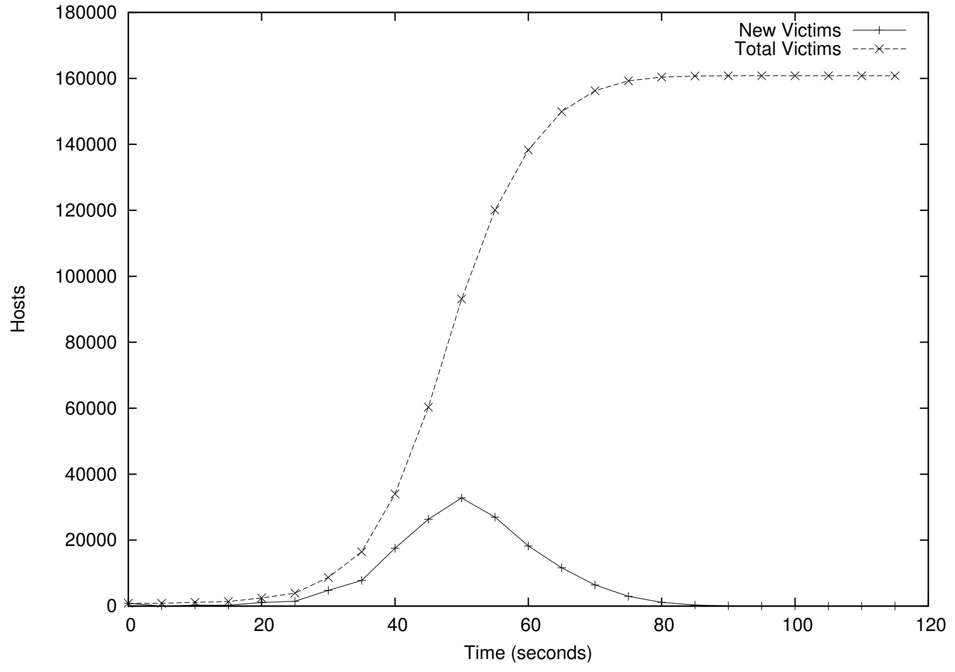

Figure 16: Advanced Multi-Threading Worm

6 Improving the simulation

6.1 Network structure

In the simulation, the network was considered to be flat, i.e. each node can reach every other node directly. No routers or other parts of the network structure were simulated, except by viewing them as non-vulnerable hosts. No backbone was assumed, and no connection problems, latency issues and especially not overload situations were considered.

All of these do affect worm propagation in reality, usually to its detriment.

However, a simulation of these conditions is far from trivial, especially if it must be automatically generated as the manual creation of a class A network structure[8] is far beyond the scope of this research.

6.2 Considering limited bandwidth

Likewise, all network connections are assumed to posses unlimited bandwidth, and no congestion or failure due to heavy load was simulated. Again, it would have been complicated and time consuming to do so, and no guarantees could have been made that the results were not dependent on the specific conditions that were generated.

The simulation did count the number of probes sent out and an estimate on the total bandwidth used, and these will be described in section 7 on page 25. They had no influence on the propagation of the worm itself, however.

6.3 Reactions and host-based effects

No effort was made to consider patching or other countermeasures by vulnerable or victim hosts. Since the goal of this research was to create a worm that would spread too fast for non-automated

[8]Or better yet: Several to ensure that the results are not tainted by a specific network topology

countermeasures, these were left out even from the initial slow algorithms in order to allow comparison without compensation for these differences.

6.4 Finer resolution

The simulation was greatly hampered by its choice of a 1-second minimum resolution. Very few worms actually require a full second to infect a host, for example, much less two. Even though the speedup of this particular change can be expected to be only about 5% (as seen in section 4.3.1 on page 15), a finer resolution in all aspects, including not only infection, but also probing and identifying non-vulnerable hosts, can be expected to increase the propagation speed.

In fact, the Sapphire/Slammer worms high speed and propagation could not be simulated without a much finer resolution, as was used in the advanced simulations.

7 Estimating the impact on the network infrastructure

Even ignoring any malicious payloads (which will be discussed in section 8), a worm of size will have a considerable impact on the network infrastructure, through the volume of scans and/or infection attempts being made.

The actual impact varies greatly depending on the propagation algorithm. Generally, better algorithms will have a more severe impact on the network infrastructure. While, for example, the initial worm of figure 1 sends out a total of about 7600 probes per second once it has reached saturation, that of figure 7 on page 13 with reduced timeouts, multi-threading and local preference, scans the network with over 160,000 probes per second. This is overshadowed by the prescan-preinfect worm of figure 12, which sends out over 500,000 probes per second after reaching saturation, not to mention the advanced multi-threading worm of figure 16 with over 6.5 mio probes per second.

Turning this data into network load requires an estimation of packet sizes for both scanning and the worm itself. Also, the total load on the entire network is less interesting than the load on local clusters. As both are outside the scope of this article, only a single example was investigated in-depth:

7.1 Example traffic graph

Assuming a worm of 2048 bytes size, with an average MTU of 1500 bytes, spreading over TCP. The worm would require an initial 3-way handshake, 2 packets to transfer itself, and shutdown. The total overhead will be 676 bytes, so each infection causes 2724 bytes of traffic. In addition, each probe will cause 296 bytes of traffic if the opposite site does not reply, or 134 bytes of traffic if it replies with an RST.

The "combined" worm shown in figure 5 would then after reaching saturation create a total (network-wide) traffic of about 5.6 GB/min, every minute[9].

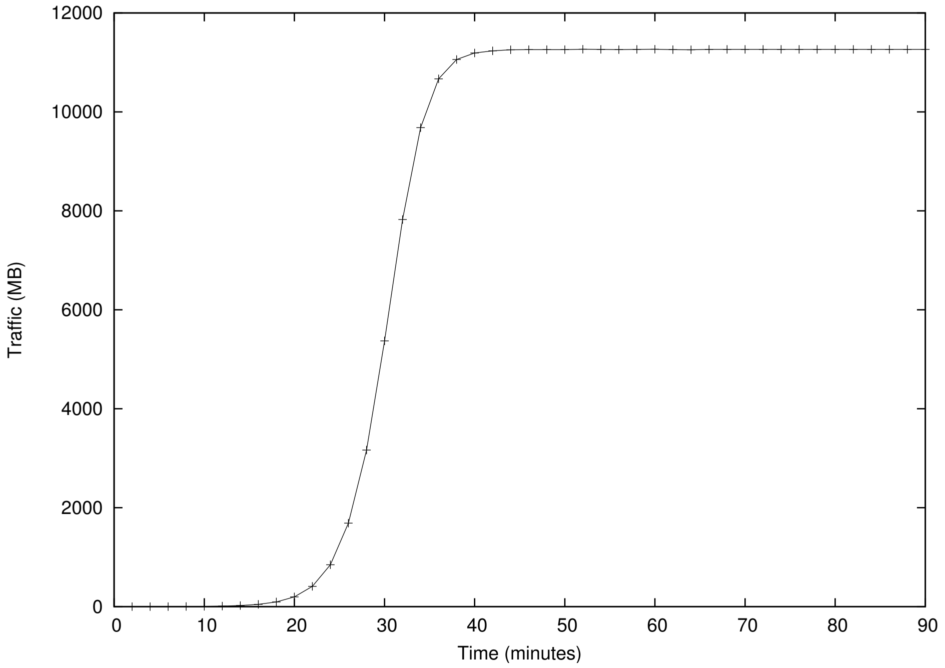

Figure 17: Traffic usage of an example worm

[9]The graph shows traffic per tick, i.e. 2 minutes.

7.2 Packet count

Traffic should not be measured only in size, but also in number of packets, as many small packets can affect a network as badly or worse than large amounts of data. Any contact with a remote system is registered as one "probe" by the simulation software. No effort was made to translate probe count into actual packet count, as low-level network details would have to be simulated to do this reliably. Much also depends on implementation details, e.g. how many re-transmits would a scanning algorithm attempt?

One reasonable assumption would be to set each probe equal to a about 2 packets. For an infection or infection attempt under the data above, 8 packets would be needed, but the majority of probes, those to offline system or systems not running the vulnerable service, 2 packets would be sufficient. This includes one retransmit for the scanning, or the RST or ICMP packet in return to a closed port.

7.3 Effects of propagation algorithms

Taking only the peak values into account, various propagation algorithms show clear differences in how much traffic they cause, as shown in table 3. All data was normalised to one minute intervals, so various algorithms can be compared.

Algorithm	Peak Traffic	Peak Probes	Peak reached	Reference
random propagation	0.3 GB/min	0.5 mio/min	ca. 620 min.	page 7
random, multi-threading	2.7 GB/min	4.6 mio/min	ca. 75 min	page 10
combined	5.6 GB/min	9.4 mio/min	ca. 45 min	page 11
combined w/local pref	6.9 GB/min	10.0 mio/min	ca. 40 min	page 13
sequential	0.3 GB/min	0.5 mio/min	ca. 940 min	page 14
pre-scanning	7.7 GB/min	17.4 mio/min	ca. 20 min	page 17
advanced scanning	28.8 GB/min	59 mio/min	ca. 6 min	page 22
advanced, multi-threading	1093 GB/min	400 mio/min	ca. 80 sec.	page 23

Table 3: Traffic usage of various worms

It can now be considered proven that massive activity is the key to worm propagation. Stealth and speed are mutually exclusive. It has also been shown *which* approaches to the problem offer the highest increases in propagation speed.

8 Payloads

While this paper is not about payloads, the discussion above opens up new options in this area. It has so far been generally believed that a destructive worm, e.g. one that wipes the hard-disk of its host, would not live long or spread far because it destroys the very population it thrives on.

This is not entirely true. If the worm can estimate how close it is to saturation, it can very well wait with its destructive intent until saturation has been reached or is near, because all vulnerable machines have been infected and there is nowhere else to go anyway.

The payload need not be targeted at the host, however. Several past worms had DDoS attack payloads. They were keyed to commence their attacks at a certain point in time, giving their victims ample time to prepare. If a worm were able to dynamically calculate its position in the growth curve, it could unleash its attack at an a) not immediately obvious and b) optimal point in time.

It seems obvious to the author that pointing a DDoS network of this size at a single site is ludicrous overkill. Even in the simulated network, and with a limit of 56 KB/sec per host, the worm would command almost 9 GB/sec. Scaling the numbers up to Internet size, the DDoS would be on the order of 2 TB/sec, most of which would never reach the target due to network overload in the local or backbone routers. Not to mention that many of the DDoS zombies will have much more bandwidth available then the conservative 56 KB/sec.

8.1 Propagation with destructive payloads

In order to verify the assumption made above, a simulation was run with a destructive payload. The worm in figure 18 shows a variant of the advanced multi-threading worm from page 23 which starts destroying hosts 60 seconds after it starts spreading. The chance that it destroys its hosts instead of spreading further is 5% per tick (5 seconds) afterwards, i.e. at 70 seconds it is 10%, at 80 seconds it is 20%, etc., with a maximum of 50% after second 110.

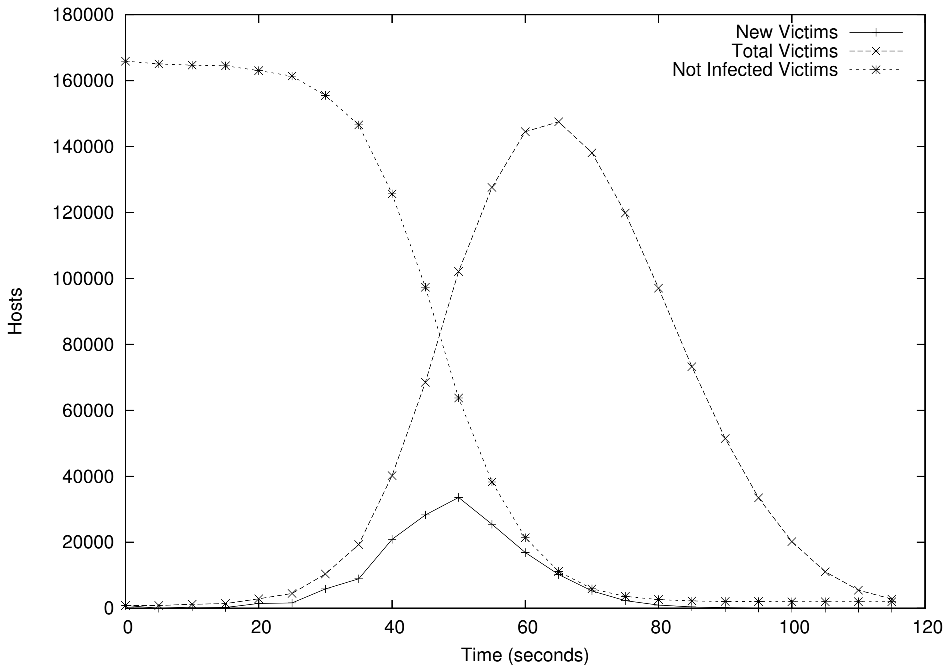

Figure 18: Advanced Worm with destructive payload

The propagation speed is not affected very much by the destruction the worm brings to its hosts. An additional curve has been plotted in this graph, showing the number of vulnerable, but not infected systems. As it clearly shows, there are few survivors. At the end of this simulation run, there were 2.774 infected and 1.979 not infected systems left, of an initial 166.730. While this not quite "annihilation", it does mean that within two minutes, 161.977 hosts or about 97% of the vulnerable population were wiped out.

Most importantly, however, comparing these numbers to those of the advanced worm proves the point. The advanced worm also did not reach a 100% infection ratio, but left 1.978 survivors.

Scaling this simulation up to Internet size, even very carefully, shows that it is entirely feasible to destroy on the order of magnitude of 40 mio. hosts in less than 5 minutes. This assumes a weak exploit against a small minority of systems. If the exploit affects a larger part of the population, for example the exploit used by Blaster, the damage will be both quicker and much more extensive.

8.2 Local Network Destruction

Another possibility to combine even more speed advantages is available if a specific target can be attacked from the inside. Individual targets, say the network of a major corporations, usually have more and softer targets than the Internet at large.

An attack on a corporations in-house network, utilising an exploit against the workstation systems, could infect and wipe out 98% of the entire client population in one minute. This is due to the fact that in most internal networks, the clustering structure is much stronger, more of the address space is used, and most of the systems are part of a monoculture, often featuring thousands of virtually identical systems.

Several simulations were run with a Class B network size, with other parameters modified to more closely represent an internal network instead of the Internet. The worm spreads from a single entry point and starts destroying its hosts after 30 seconds. As and would reach the aforementioned 98% destruction rate easily within the first minute. As figure 19 shows, it easily accomplishes its goal, destroying the affected corporate infrastructure in well below the reaction time of even the best IT department.

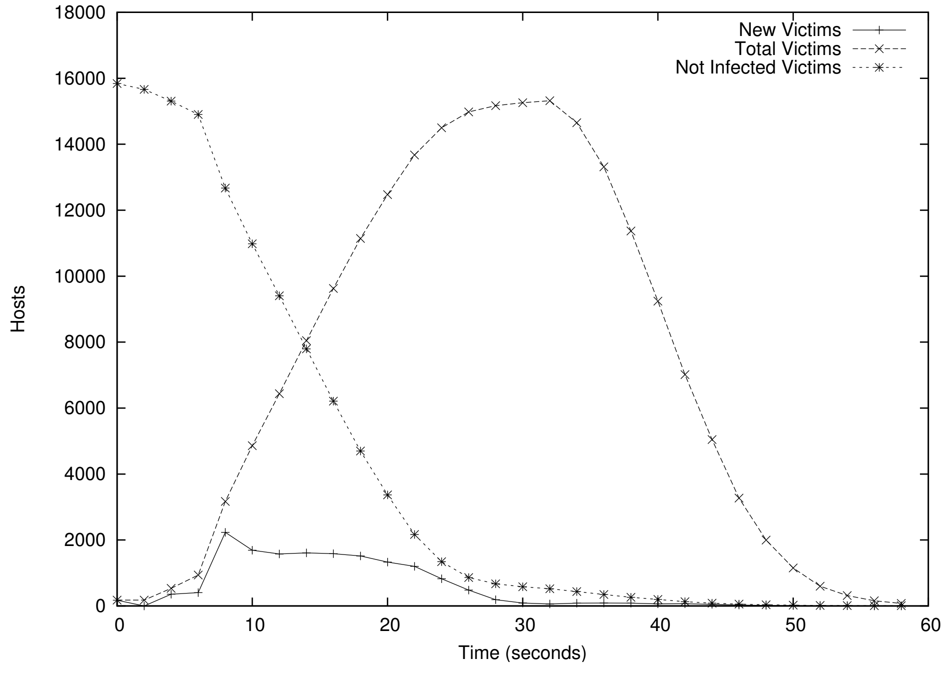

Figure 19: Local Network Destruction

8.3 Self-Preservation of Worms

Even though the worms presented will infect their host population much too fast for any kind of manual countermeasures, it is noisy enough to catch immediate attention. If its purpose requires the worm to survive for more than a few hours, it will need some way to prevent or at least delay countermeasures such as patching.

Research into this area is currently being conducted. Some worms seen in the wild already disable anti-virus software or take other primitive baby steps into this very direction.

9 Conclusion

It has long been argued that future worms will be stronger and faster. This research shows that Flash Worms are feasible without unrealistic assumptions about their properties, and which elements make the main difference in designing such a worm:

- Some way to discard the vast amounts of unused or offline addresses very quickly
- An initial pre-infection step, or a targeted distribution into known-vulnerable networks
- Multi-threading

As the primary means of improving propagation speed, as well as:

- Identification of already infected or non-vulnerable hosts
- Fast infection of victims

As additional options to gain further speed improvements.

From these points it follows that the addition of a stateless scanning algorithm, optionally with an additional identification step such as a banner scan or OS fingerprint, will be the next major leap forward in worm development.

It should also be expected that worms will carry a small data segment that contains such information as to which large networks are unassigned or unused and can be skipped (this was discussed in section 4.7 on page 19).

9.1 Countermeasures

The main lesson consists in the knowledge that any kind of advanced worm will be too quick to allow for manual analysis and countermeasures. Even the preparations for a typical analysis take longer then the advanced worms presented here need to reach saturation[10]. By the time countermeasures are taken, the worm had more than ample opportunity to accomplish whatever it was designed to accomplish.

9.1.1 Preventive Countermeasures

One defence could be constructed by having massive scanning and/or incoming connections, as well as other typical footprints of worm attacks trigger automated self-defence mechanisms. Some systems of this kind are already deployed or under development as DoS and DDoS defences.

Contrary to popular opinion and - otherwise valid - arguments about mono-culture, reducing the number of vulnerable systems only offers a marginal slow down effect. A simulation run with only a quarter the vulnerable hosts (about 41,000) reached saturation in about 120 seconds. For the forseable future, it is certainly safe to assume that exploits will be found that 1% or more of the Internet connected hosts are vulnerable to.

9.1.2 Legal Countermeasures

Improving the law enforcement agencies abilities to track and incarcerate worm authors might serve as a deterrence and thus reduce the likelihood of an advanced worm being actually deployed.

However, the past record in this area is not very promising, and the issue of jurisdiction is not going to be resolved quickly. In addition, other crimes and their penalties show that even the harshest sentence does not deteredeter everyone. For an advanced worm, a single unimpressed culprit would be enough.

[10]This point remains true, even if the complications not taken into account, such as bandwidth limitations, slow the worm down. Unless the slowdown factor is very large, in the magnitude of 10 or more, it will not invalidate the point that manual intervention is too slow.

It is not likely that harsher sentences, stronger surveillance or better educated law enforcement personal would make a measurable difference in global security.

9.1.3 Local Protection

For short and medium time security planning, therefore, the main focus should be to protect the local network from a hostile Internet. Current worms could be filtered easily on border routers, provided those have enough spare computing power. However, if it becomes necessary, it is likely that worms will inherit polymorphic capabilities from viruses. These would defeat any simple filters, and routers are not suited for more advanced virus scanning methods, even if the problem of false positives could be ignored.

9.1.4 Conclusion

I am thus forced to conclude this paper without a good recommendation for global countermeasures, as an in-depth discussion of these is outside its scope.

The results of my research do show some of the parameters for the yet-to-be-developed countermeasure systems, however, as well as the kind of worms that can be expected to visit the Internet sometime soon.

In the past, worm research was usually ahead of actual worms found in the wild, and in fact researchers were regularly underwhelmed by the new worms that appeared on the Internet. A prognosis of doom is certainly not adequate, and even if a worm of destructive potential were to appear, viewed from a long-term perspective, its effect on the Internet could be argued as beneficial, due to the reduction of vulnerable hosts. Whether or not principles of evolution hold true for this artificial environment, however, remains to be seen.

10 Appendix

About the author

Tom Vogt is an independent security researcher in Hamburg, Germany. He is currently employed as a systems analyst for a local telecommunications company, but is privately researching various security topics. He has given numerous presentations and training courses on security issues and related topics.

References

[1] As detailed in http://arbornetworks.com/downloads/dark_address_space.pdf

[2] http://www.caida.org/analysis/security/code-red/coderedv2_analysis.xml

[3] http://www.sans.org/rr/paper.php?id=93

[4] http://abcnews.go.com/sections/scitech/DailyNews/codered2_010806.html,
http://www.osopinion.com/perl/story/12546.html and others

[5] http://aris.securityfocus.com/alerts/codered2/

[6] http://www.icir.org/vern/papers/cdc-usenix-sec02/

[7] http://www.crimelabs.net/docs/worm.html

[8] http://www.sans.org/rr/catindex.php?cat_id=36

[9] Thanks to Armin Krack <armin.krack@nruns.com> for his help with the math in this part

[10] http://www.caida.org/outreach/papers/2003/sapphire/sapphire.html

[11] http://www.sans.org/rr/paper.php?id=88

[12] http://www.doxpara.com/read.php/code/paketto.html

[13] "Stealing the Network", Syngress Publishing, 2003. An overview of scanrand including this number is on page 225.

[14] http://www.usenix.org/publications/login/2003-04/openpdfs/motd.pdf