Experiences with Worm Propagation Simulations

Arno Wagner
wagner@tik.ee.ethz.ch

Thomas Dübendorfer
duebendorfer@tik.ee.ethz.ch

Bernhard Plattner
plattner@tik.ee.ethz.ch

Roman Hiestand
romhiest@ee.ethz.ch

Computer Engineering and Networks Laboratory (TIK)
Swiss Federal Institute of Technology Zurich
ETH-Zentrum, CH-8092 Zurich

ABSTRACT

Fast Internet worms are a relatively new threat to Internet infrastructure and hosts. We discuss motivation and possibilities to study the behaviour of such worms and degrees of freedom that worm writers have. To facilitate the study of fast worms we have designed a simulator. We describe the design of this simulator and discuss practical experiences we have made with it and compare observation of past worms with simulated behaviour. One specific feature of the simulator is that the Internet model used can represent network bandwidth and latency constraints.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Invasive software (e.g., viruses, worms, Trojan horses)*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Invasive software (e.g., viruses, worms, Trojan horses)*; I.6 [**Computing Methodologies**]: Simulation and Modeling

General Terms

Security

Keywords

Internet Worms, Simulation, Bandwidth, Latency

1. INTRODUCTION

Internet worms, such as *Code Red* [2] or more recently the *Sapphire* [3] are an emerging threat to Internet hosts and infrastructure. These worms can compromise very large numbers of hosts in a short time, down to a few minutes [16]. The compromised hosts can then be used to perform other attacks, like massively Distributed Denial-of-Service attacks [9, 4].

Study of the behaviour of a worm during its propagation phase is important for several reasons. One is the creation of early warning systems that can detect a propagating worm, and in an ideal case also can give preliminary propagation analysis and perhaps even a captured specimen. Today such systems do not exist and significant work will be needed to turn them into a reality. Another interesting aspect is threat analysis with regard to speed and number of hosts a worm can compromise in a specific time frame. Prediction and analysis of collateral damage, e.g. ARP-floods in badly configured subnets, is also an aspect that needs to be looked into.

Since worms observed in the past cover only a small portion of the possible parameter space, it is important for countermeasure design to be able to roughly predict characteristics a worm can have.

We believe that simulation is a key tool here and discuss experiences made and insights gained with a prototypical simulation tool for worm propagation.

The paper structure is as follows. We start with a more detailed discussion of the motivations why worm behaviour prediction is important in Section 2. Section 3 goes more into the possibilities how worm propagation can be studied. Section 4 examines the relevant parameter-space and how these parameters relate to worm behaviour. In Section 5 we present the implemented simulator and give insights into experiences made with it in Section 6. Finally we briefly discuss the effects of IPv6 in Section 7 and give an overview of related work in Section 8. Section 9 finishes the paper with a conclusion.

2. WHY PREDICT WORM BEHAVIOUR?

The benefits of predicting worm behaviour are numerous:

- Better understanding of the behaviour of worms observed in the past
- Estimations of a worm's threat potential
- Estimations of the impact of future worms on the Internet

WORM'03, October 27, 2003, Washington, DC, USA.

- Basis of the design of detection mechanisms for worm spreading

- Determination of parameters relevant for worm characterisation

2.1 Traffic Prediction

Traffic prediction for the worm spreading phase helps to estimate the decrease in performance of an affected network. Slow spreading worms might not even be visible in traffic monitoring tools as they are well hidden in regular traffic variations. However, if specific characteristics of a worm are known, a detection might still be possible.

2.2 Speed Prediction

The Sapphire worm infected more than 90% of all vulnerable hosts in the Internet within 10 minutes [10]. Since manual intervention is too slow to deal with this, there is a need for semi- or full-automatic tools that detect and analyse a spreading worm and activate countermeasures in near real-time.

2.3 Threat Evaluation

Given that modern worms have the potential to infect most vulnerable hosts in the Internet within a short time, these worms pose a real threat to the Internet infrastructure. It is important to determine what the possibilities and limitations of this attack tool are to concentrate countermeasure efforts in the most vulnerable places.

3. SIMULATION AND ALTERNATIVES

We will now discuss different ways to study the characteristics of a piece of self-propagating code.

3.1 Mathematical Models

The most powerful approach is probably the creation of a realistic mathematical model that allows behaviour prediction in a closed form, i.e. with no or very little iteration. The problem with this approach is that such models are not generally available and are usually hard or even impossible to create.

3.2 Testbeds

Testbeds allow to actually set free some self-replicating code in an isolated and limited environment and to observe its behaviour. The most obvious limit of a testbed is that it cannot be created in a size approaching the size of the Internet. Another serious problem is that a testbed needs to use real self-propagating code, which is difficult to obtain. There are also legal and moral problems with creating and handling such code.

3.3 Real World "Experiments"

If a testbed is too limited, why not use the Internet itself? While worm code authors certainly take this freedom, this is not an option for scientific study because of the damage being done. In a very limited sense the use of the whole Internet is possible, namely in observing the behaviour of worms that have been set free by people not hampered by ethical considerations. We have observation equipment in place in a moderately sized backbone network to observe the next Internet outbreaks. See Section 8 for further details.

3.4 Simulation

In a sense simulation is a mathematical model in which some of the functions used rely heavily on iteration. In order to reduce computational complexity abstraction and approximation of the inner mechanisms of the object studied is often used. This allows computation of functions that are not well understood in a mathematical sense. The analytical approach of mathematical modelling is replaced with an experimental approach, in which scenarios are simulated and then analysed. Simulation is often a very effective tool to understand complex processes.

A significant drawback of simulation is that due to abstraction the simulation results can differ significantly from real behaviour of the system under study. A way to verify and optimise simulation accuracy is to simulate events that have been observed in the real system and compare simulation outputs to the measured data.

4. WORM CHARACTERISTICS

There is significant freedom in worm design. For this paper we do not care about the specific vulnerabilities exploited for system compromise, but only about time, bandwidth and transport protocol used by a worm.

A worm writer basically implements the following process:

1. Identify a vulnerable host

2. Compromise the target host

3. Transfer the worm and activate it

For some vulnerabilities all these steps can be combined into a single network packet, as was done in the case of the Sapphire. For others, the steps have to be done separately.

We believe that for the study of worm propagation a very abstract view of these steps is sufficient. Steps 2 and 3 can be modelled as exchange of a specific amount of data with a specific protocol and optional time delay. Step 1 is a little more complicated, but can still be modelled disregarding vulnerability details.

TCP vs. UDP

The main choice in the transport protocol is whether it is connection-oriented or not, for simplicity represented by TCP and UDP. For worms that infect a distributed application, like a P2P system, other models might be needed [17]. The protocol used is usually directly determined by the vulnerability that is exploited by the worm.

For the case of UDP, resource consumption in the attacking host is small. A typical scenario is to send out UDP packets to random hosts, while keeping very little state information for each target, or none at all if the attack can be executed by sending a single UDP packet. Disadvantages are that the size of a UDP packet is constrained to around 50 kiB[1] and data packets with a payload larger than 1472 Bytes will be transported using IP-fragmentation.

Use of TCP incurs penalties for connection establishment, timeouts and error handling. On the plus side there is no data size limit. The most significant disadvantage of TCP is that a connection attempt to a non-existing host fails only

[1]This is OS dependent. We found that e.g. Solaris has a limit around 50 kiB, Linux a little higher. 64 kiB is the definite protocol limit.

after a long timeout and consumes OS resources until it does. There are ways around this, but they require that the worm implements its own modified version of TCP, which makes worm design more difficult and increases worm size.

Amount of Data Transferred

The time a worm needs to propagate after a vulnerable target has been identified depends mainly on worm size and available bandwidth. Additional delays may be present, e.g. if a reboot of the attacked host is needed. Data transfers form a specific signature of a worm and can be used for detection purposes. Obviously, a large worm will generally propagate significantly slower and far more visible, so worm writers will often aim to write small worms.

Scanning Strategy

The scanning strategy is the method used to select the next host to be probed. One end of the spectrum is *random scanning*, which selects a next IP address at random. This is one of the most primitive strategies. Surprisingly, worm authors have often failed to write clean random scanners [10, 2]. Mistakes include constant PRNG seeding after propagation and use of inferior PRNGs with non-even value distribution. Random scanning has also been implemented with preference for addresses in the same subnet and other modifications.

On the other end of the spectrum is partial or full predetermination of the target sequence. The worm operator does a stealthy reconnaissance creating a list of vulnerable hosts. The list of targets, called *hitlist*, is then added to the worm. Each new copy of the worm created in propagation then gets a part of the remaining hitlist to work on.

Hitlist scanning can be used to speed up the initial slow phase of worm propagation. It does not work so well in later phases, when a lot of infected hosts are active. It is possible to combine an initial propagation mechanism based on a moderately-sized hitlist with a random scanning method that takes over when the hitlist has been processed. The main disadvantage of using hitlists is that their creation might be detected, leading to countermeasures before the worm is set free.

Latency vs. Bandwidth Limit

Even though Code Red I and Sapphire both used random scanning, their propagation speed was different by several orders of magnitude. The number of Sapphire infected hosts doubled initially every 8.5 seconds while the Code Red Iv2 worm population had an initial doubling time of about 37 minutes [10]. The reason for this difference lies in the choice of the transport protocol and in the size of the transferred worm code.

Sapphire uses a single UDP packet with a total size of 404 bytes. Since there is no connection establishment with UDP, the spreading speed is mostly independent of latency but strongly dependent on bandwidth. An infected host can send as many infection packets as its network link and protocol stack allow.

Code Red uses TCP, which implies the use of a three way handshake for connection establishment. Consequentially latency is the main limit on propagation speed. In addition OS constraints limit the number of parallel connection attempts that can be made. Latency limited worms can also become bandwidth limited when their scanning traffic exceeds network resources. For Code Red this happened after about 15 hours.

5. SIMULATOR DESIGN

The simulator is started from the command line under X11. It was developed under Linux, but should run under most Unix-like operating systems without modification. It first reads the parameter values and then opens two plot windows. The *speed plot* shows the number of infected hosts vs. time and the *traffic plot* shows the total scanning and infection traffic vs. time. Plain text output is also available. The simulator code is available upon request to the authors.

5.1 Simulator Structure

Our aim was to create a modular and flexible simulator that can easily be extended. We chose the scripting language Perl as basis for the implementation, since it is well suited for rapid prototyping and is fast enough for our purposes as our evaluation in 6.2 shows. Perl modules are used to structure the code and to facilitate extensions. Plotting is done with gnuplot. A pipe is kept open to each instance of gnuplot and automatically[2] flushed to generate an updated plot when the simulator has finished a number of iteration steps.

5.2 Internet Model

The Internet model is at the very core of our simulator. We were looking for a model that is complex enough to represent prevalent characteristics of today's Internet. At the same time it had to be simple enough to enable efficient simulations. Inspired by the Napster and Gnutella P2P client connection measurements in [15] we chose a model that abstracts from single hosts.

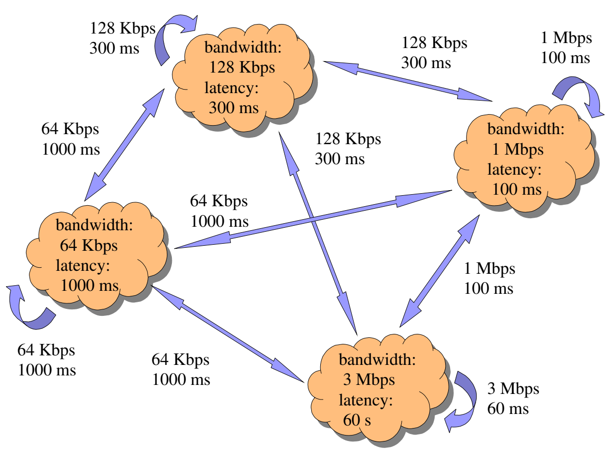

Figure 1: Example for a configuration of our Internet model

Our model divides the Internet into n different groups of hosts that belong to sub-networks with similar characteristics. Each host group has two defining parameters: *bandwidth* and *latency*. The bandwidth and latency of a connection between any two groups are chosen as the minimum bandwidth and maximum latency of the groups. Figure 1 shows a 4-group configuration of the Internet model that is

[2]This can be done in Perl by using `select(G); $| = 1;`, with `G` being the handle of the pipe.

used in our simulator. Details of the host distribution can be found in Table 1. We also specified a 10-group configuration, given in Table 2. The average bandwidth per host in Table 1 is 1157 kbit/s for Napster and 1544 kbit/s for Gnutella, for Table 2 it is 1176 kbit/s.

The given percentages are measurements from [15] and assume that the user population of Napster and Gnutella are representative for the whole Internet. The differences between the Napster and the Gnutella numbers show that this approach is not very accurate. Still these are the best figures we were able to find.

The details of the underlying measurements as well as more information on host characteristics in the Napster and Gnutella P2P filesharing systems can be found in [15]. For continued usefulness of the model and the simulator these numbers will have to be updated from time to time.

Bandwidth	Napster	Gnutella	Latency
64 kbit/s	32%	10%	1,000 ms
128 kbit/s	5%	14%	300 ms
1 Mbit/s	38%	38%	100 ms
3 Mbit/s	25%	38%	60 ms

Table 1: Internet models with 4 groups

Our Internet model turned out to be powerful enough to simulate many cases of worm behaviour. Still for some cases modifications were needed to get realistic results, as discussed in Section 6.

The model could easily be extended to support asymmetric connections in order to simulate ADSL or Cable modem connections that e.g. in some European countries have a downstream speed which is two to four times faster than the upstream speed. Also, the TCP slow start behaviour is not modelled. However, as most worms are rather of small size, it could be represented by choosing a lower bandwidth than the actually available bandwidth.

The nature of our Internet model is well suited for a quantitative analysis of worm spreading, however it is not suited for traffic prediction for a specific host.

Bandwidth	Napster	Latency
14.4 kbit/s	4%	1000 ms
28.8 kbit/s	1%	1000 ms
33.6 kbit/s	1%	1000 ms
56 kbit/s	23%	1000 ms
64 kbit/s	3%	1000 ms
128 kbit/s	2%	300 ms
256 kbit/s	44%	300 ms
512 kbit/s	14%	100 ms
1.544 Mbit/s	5%	60 ms
44.736 Mbit/s	2%	60 ms

Table 2: Internet model with 10 groups

5.3 Implemented Worm Parameters

Table 3 provides an overview of all worm and Internet parameters implemented by our simulator and gives their value range.

5.4 Implemented Scanning Strategies

The simulator implements three different scanning strategies, namely *Random Scanning* with even distribution, *Hitlist Scanning* with a user-defined hitlist and *Local Forced Scanning* that scans local IP addresses with a higher rate than remote addresses.

The effect of hosts being already infected during worm spreading is taken into account by reducing the success probability of an infection attempt:

$$P(\text{infect}) := \frac{|\text{vulnerable hosts}| - |\text{infected hosts}|}{|\text{all hosts}|} \qquad (1)$$

For each time step the simulator sums up the *infection probabilities*, as defined in (1), for each host scanned to determine the number of newly infected hosts. An error is introduced here because two scanning hosts could select the same target in a time step. This error is small as long as the number of vulnerable hosts is significantly lower than the number of all hosts. Therefore (1) is presently used in the simulator.

5.5 Output and Reporting

The simulator produces a text file that describes all parameter values for the simulation, as well as numeric data files suitable for Gnuplot input. In addition the graphical plots are displayed and updated on the screen while the simulation is in progress.

Traffic: The traffic plot shows the total traffic generated by the scanning and propagation of the simulated worm over time.

Spreading Speed: The spreading speed plot shows the total number of infected hosts over time.

5.6 Simulator Limitations

The simulator assumes an even distribution of the vulnerable hosts over the different speed groups. The Code Red worms attacked many installations of the IIS web server with the owners of the hosts not even aware they were running a web server, because IIS had been installed as part of other software packages. Accordingly the vulnerable hosts were pretty evenly distributed over all speed groups. However if a worm targets an application that is only installed on hosts that are specifically designated as servers, the vulnerable hosts will tend to be in the faster groups.

Countermeasures by network and host operators are not modelled in the simulator. The effects of such countermeasures will vary heavily depending on human behaviour and technical parameters and hence can hardly be modeled reliably.

6. PRACTICAL EXPERIENCES

In order to test and validate the simulator design and implementation, we did a number of practical experiments. Most are centered around the Code Red and Sapphire worms and variants of these, since they are probably the best understood fast worms that have been observed in the wild.

6.1 Impact of Internet Model

The Internet model serves as an approximation of the real Internet. Since precise overall Internet bandwidth and latency figures are not available, the model also serves as a method to estimate bandwidth and latency based on a limited observation of these characteristics in real distributed

Worm parameter	Unit	Lower limit	Upper limit
Hosts in the Internet	hosts	1	4,294,967,296
Vulnerable hosts	hosts	1	hosts in the Internet
Start population	hosts	1	vulnerable hosts
Simulation time span	seconds	0	no limit
Transport protocol	TCP or UDP	–	–
TCP resend on timeout	enable/disable	–	–
TCP timeout	milliseconds	0	no limit
Worm size (without header)	bytes	0	65535
Parallel scans (TCP) or scans per second (UDP)	–	0	no limit
Additional time to infect a host	milliseconds	0	no limit
Hitlist	enable/disable	–	–
Hitlist length	hosts	0	hosts in the Internet
Probability a hitlisted host is vulnerable	-	0%	100%

Table 3: Simulation Parameters

Internet applications, in our case P2P filesharing. It turned out that the Internet models needed to be adjusted to some degree to get realistic simulation results.

Sapphire

Sapphire is bandwidth-limited. Its propagation speed is roughly linear with the bandwidth directly available to the already infected hosts. When a high number of hosts has been infected, there can also be additional limitations because of ISP and backbone bandwidth limits. We assume a vulnerable population of 75,000 hosts.

Figure 2 shows a simulation graph obtained with the 10-group Internet model from Table 2. The initially infected population was 100 hosts distributed over the different speeds according to group size. The simulation deviates significantly from the observed propagation speed of the Sapphire in [10], likely because the Internet has gotten faster since the Napster measurements were taken. If the 100 initially

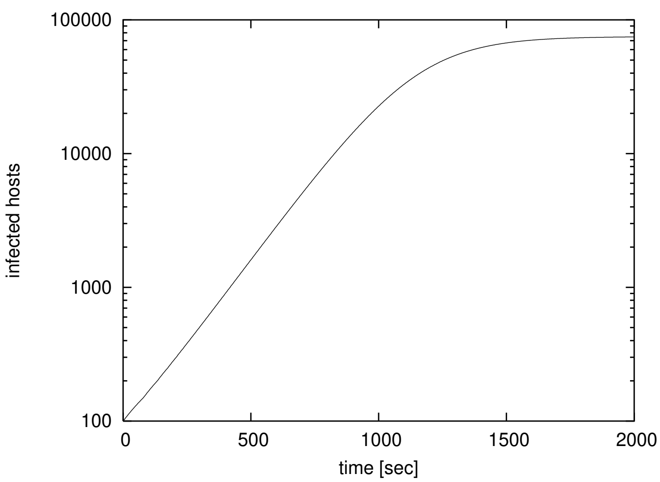

Figure 2: Sapphire: Infection Speed

infected hosts are chosen from the fastest group and in addition the fastest group is enlarged to 10% (taking evenly from the other groups) the initial doubling time is about 6 seconds and the scanning rate after 3 minutes is about 50 million per second, giving a very rough approximation for the observed Sapphire behaviour. The simulation then reaches an infection level of 90% after about 275 seconds. Figures 3 and 4 show the resulting infection and traffic plots. It can be seen that the lack of fast hosts cause the propagation speed to be sub-exponential.

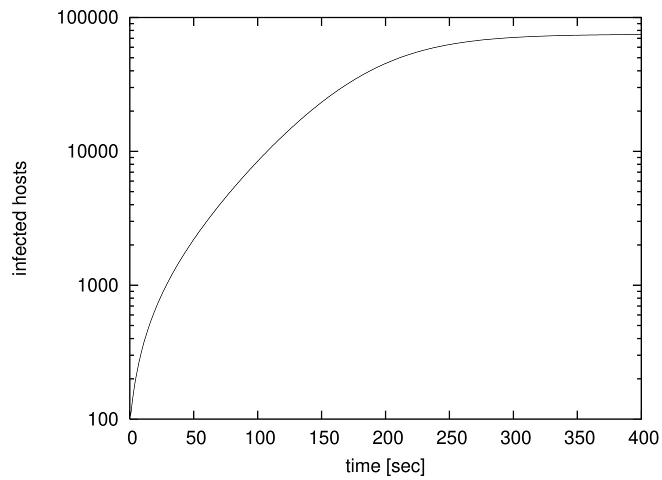

Figure 3: Sapphire: Speed with adjusted model

From these experiments we conclude that the initially infected population (obtained via hitlist or pre-infection) while critical for the propagation speed, need not be large. 100 fast vulnerable hosts are probably easy to find. From there on plain random scanning is quite effective.

To demonstrate the possibilities of the simulator, we give some more examples. The parameters are the same as for the second simulation above. Figure 5 shows Sapphire with 15,000 vulnerable hosts. The worm now needs about 1030 seconds for a 90% infection degree. This demonstrates that UDP worms with random scanning can still be used for relatively small vulnerable populations.

Figure 6 demonstrates the effect of an infection latency, for example a reboot after infection, here chosen to be 100 seconds. Infection of 90% of the vulnerable hosts now takes about 660 seconds, which shows that even with a significant infection latency the worm is still quite fast.

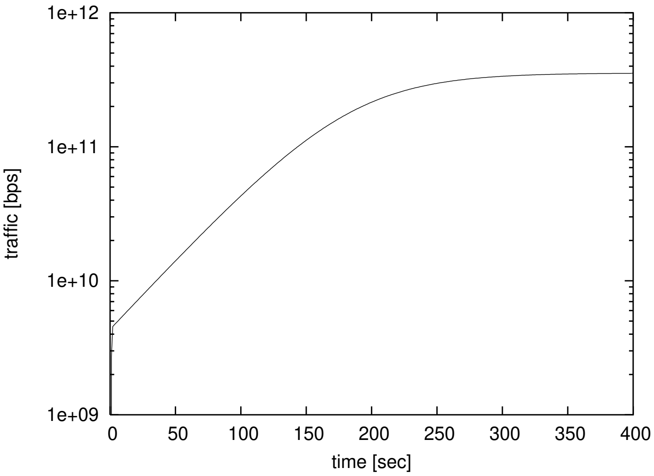

Figure 4: Sapphire: Traffic with adjusted model

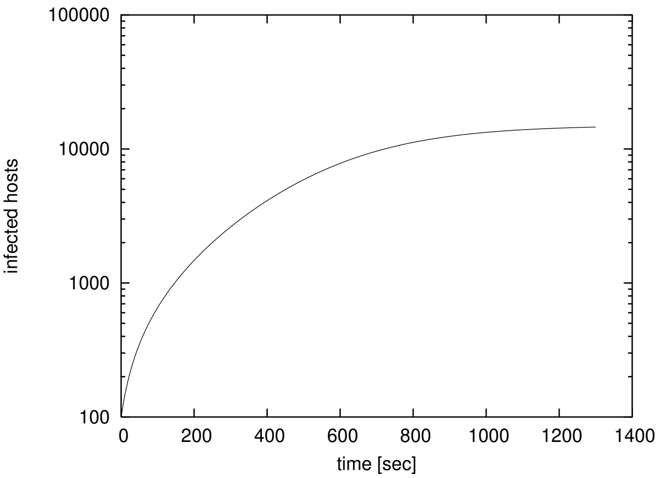

Figure 5: Sapphire: 15,000 vulnerable hosts

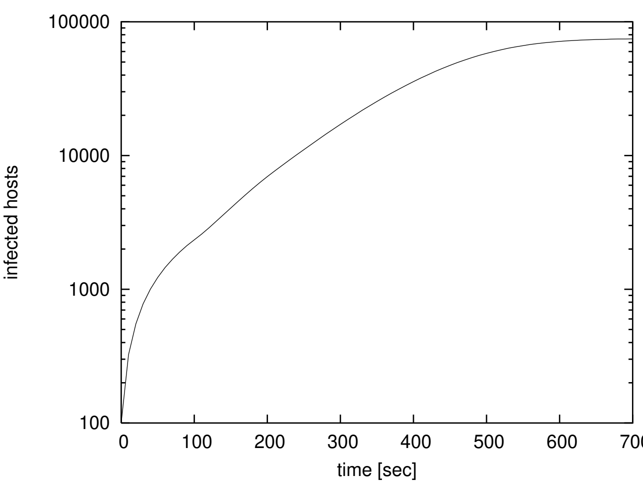

Figure 6: Sapphire: 100 sec. infection latency

Code Red

To validate our simulator's results for TCP-based worms, we tried to approximate the behaviour of Code Red Iv2. Therefore we combined data from different analyses in order to choose the most accurate parameters for our simulation. The plot by CAIDA [11] as shown in Figure 7 was used as a reference to estimate the simulator's accuracy.

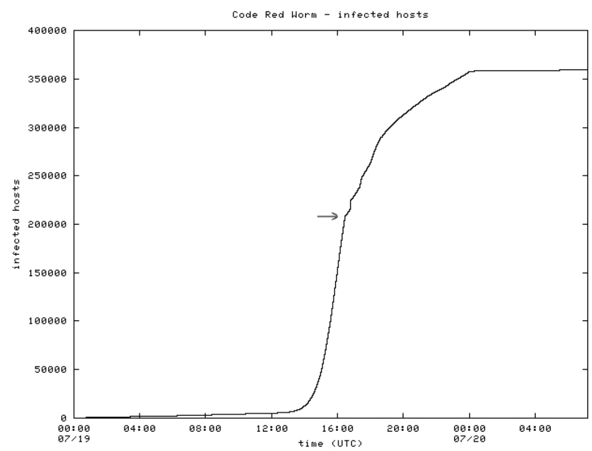

Figure 7: Code Red Iv2: Measurements of infected hosts by CAIDA

For the simulation we assumed 360,000 vulnerable hosts (7). The TCP timeout of CodeRed Iv2 was set to 21 seconds ([20]) and the number of parallel threads sending out scanning packets was set to 100 ([13]). TCP resending was disabled and a time step of 1s for the simulation was defined.

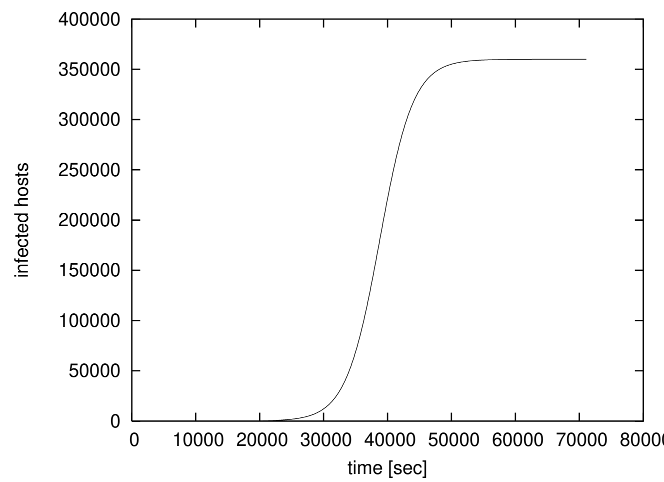

Figure 8: Code Red Iv2: Infection speed simulation matches measurements

Our results for the number of infected hosts against time using the 4-group (Napster-based) model are shown in Figure 8 and closely matches the reference plot. The plot does not show the effects of countermeasures put into place by network and host administrators that are present in CAIDA's plot. The arrow in Figure 7 shows where these countermeasures begin to affect the worm's spreading.

The logscale plot in Figure 9 shows nicely the exponential increase in he number infected hosts.

Finally in the traffic log as shown in Figure 10, it can be observed that at the saturation level of 360,000 infected

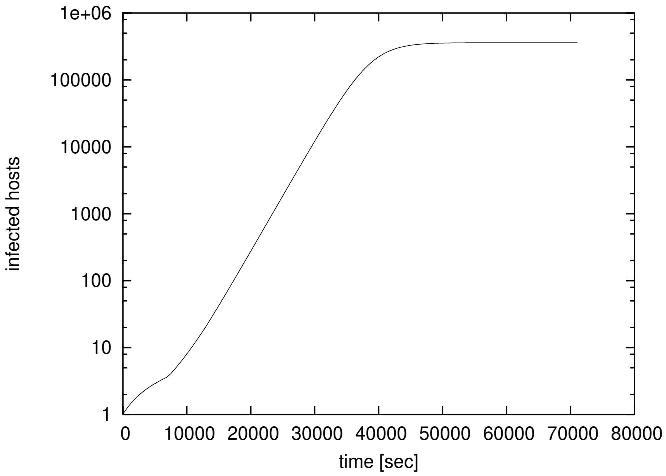

Figure 9: Code Red Iv2: Infection speed simulation matches measurements

hosts, a traffic of roughly 0.5 GBit/s is generated. Each host accounts for roughly 1.5 kbit/s as 100 parallel threads on each host send TCP SYN packets within each 21 seconds timeout interval. The fluctuations in the traffic shaping stems partially from a high synchronisation of the hosts due to a fixed time reference for all hosts in our simulator.

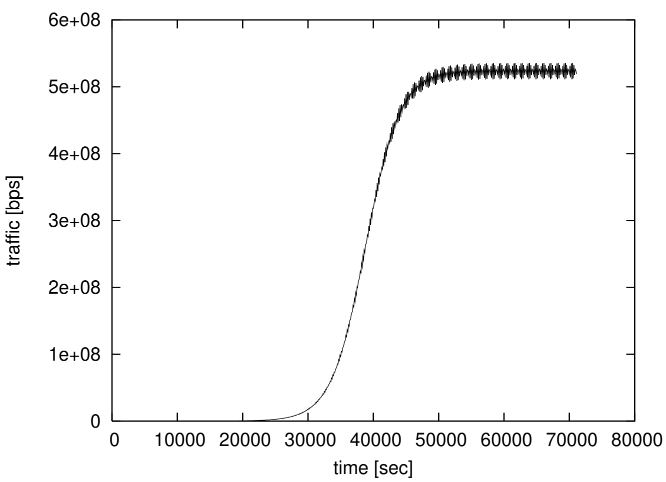

Figure 10: Code Red Iv2: Traffic simulation

6.1.1 *Parameter Variations*

A simulation of CodeRed Iv2 with the 10-group model showed only negligible differences to the 4-group case. A decrease of CodeRed's worm size to the size of Slammer showed only a slight decrease in the generated traffic. This is not surprising as the rather large worm code only propagates to vulnerable hosts and hence most of the traffic is caused by scanning other hosts.

6.2 Simulator Performance

Simulator performance varies widely with the input parameters. We did most of our experiments on an AthlonXP 2200+ under Linux 2.4., with a time resolution of 50ms for the UDP simulations and 1 second for the TCP simulations.

For UDP simulations we have observed a simulation runtime of below 20% simulated time for a 4 group model. With a 10-group model overall simulation time was still lower than simulated time in many cases. A high infection latency time is the one factor that slows down things massively, since more state has to be kept. Reducing the time resolution drastically speeds things up, which allows to balance accuracy against performance. We believe that simulator performance is good enough for many applications.

The Perl-based implementation allows easy modification if a need to simulate additional effects arises. We feel that this flexibility is more important than the benefits of a faster implementation with a compiled language. However the simulator presented here is not suitable for a simulation with a large number of host groups, that e.g. model individual subnets. But we are not aware of speed and topology statistics that could be used as basis of such a model. Even if they were available, we think that Internet topology is too unstable for such a detailed model to stay usable without frequent and possibly costly updates.

7. IPV6 AND RANDOM SCANNING

IPv6 offers a 128 bit address space [14, 8]. It is not quite clear how much structure will be contained in addresses actually assigned to hosts in the future. For example only 1/8 of the address space is currently assigned to global unicast addresses. Furthermore 64 bits may be used for interface identification. In case of an 48 bit MAC address, there are significantly less than 48 bits of randomness in these 64 bits, although the structure is not very simple. Still we expect that random scanning will be ineffective with wide deployment of IPv6. One possible way around this problem is already in use by email-based worms. The idea is to use locally known addresses as targets. Addresses can be found in local DNS caches, ARP caches, currently open connections, web browser bookmarks, contact lists of P2P systems and other places. It remains to be seen how effective such mechanisms are and whether worms can achieve fast propagation speeds under IPv6 without resorting to large hitlists.

8. RELATED WORK

Generally speaking the Internet is difficult to simulate, a good overview can be found in [7]. Still with a narrow focus the task becomes easier. In [20] there is a simulation-based analysis of the Code Red worm using methods from epidemiology.

One of the first to recognise the immense threat worms are to the Internet is N. C. Weaver who coined the term "Warhol Worm" [19] for very fast worms. This work is extended in [16].

Since worms can be used to compromise a large number of hosts, they can be used in direct preparation of massively Distributed Denial of Service attacks. There is a lot of work that shows the possibilities to fight (D)DoS attacks in end-systems or end-networks. For example, many variants of IP-traceback deal with the problem of identifying the sender of spoofed IP packets.

DDoSVax

The simulator described in this paper was created in the context of a research project called DDoSVax [6]. The project deals with detection, analysis and countermeasures for DDoS attacks, as well as worms used in preparation of such attacks. The focus is on backbone networks and consequentially on massive network events that are visible on the backbone level.

9. CONCLUSION

Understanding worms and their propagation mechanisms is a relatively new research area with significant impact on Internet stability. It is likely to grow even more important when worm writers get more experienced and outbreaks get more common. Because software engineering techniques can only mitigate part of the risks and are often not even applied to any reasonable degree, many hosts will remain vulnerable. The only solution dealing with the residual risk is for the Internet infrastructure to obtain defensive mechanisms. For the specific problems of worms, early detection is the key to any defence [12].

We have shown that simulation relying on a relatively simple Internet-model and implemented in a scripting language can provide valuable insights into worm propagation events.

One problem is that relatively few global outbreaks of fast worms have been observed so far, and the observation detail was not very good in these cases. But there are certainly more fast Internet worms to be expected. We are working on better observation equipment and we will use measurement data gathered in the future to refine and improve future versions of the simulator.

10. ACKNOWLEDGEMENTS

Part of the DDoSVax project [6] is funded by SWITCH.

11. REFERENCES

[1] P. Barford and D. Plonka. Characteristics of Network Traffic Flow Anomalies. In *ACM SIGCOMM Internet Measurement Workshop*, 2001.

[2] CAIDA. CAIDA Analysis of Code-Red. `http://www.caida.org/analysis/security/code-red/`. visited June, 2003.

[3] CERT. CERT Advisory CA-2003-04 MS-SQL Server Worm. `http://www.cert.org/advisories/CA-2003-04.html`, 2003.

[4] R. K. C. Chang. Defending against Flooding-Based Distributed Denial-of-Service Attacks: A Tutorial. *IEEE Communications Magazine*, October 2002.

[5] R. Danyliw and A. Householder. CERT Advisory CA-2001-19 "Code Red" Worm Exploiting Buffer Overflow In IIS Indexing Service DLL. `http://www.cert.org/advisories/CA-2001-19.html`, 2001.

[6] DDoSVax. `http://www.tik.ee.ethz.ch/~ddosvax/`.

[7] S. Floyd and V. Paxson. Difficulties in Simulating the Internet. *IEEE/ACM Transactions on Networking*, 2001.

[8] `http://www.ipv6.org/`.

[9] J. Mirkovic, J. Martin, and P. Reiher. A Taxonomy of DDoS Attacks and DDoS Defense Mechanisms. `http://www.lasr.cs.ucla.edu/ddos/ucla_tech_report_020018.pdf`, 2002.

[10] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the slammer worm. *IEEE Security and Privacy*, 4(1):33–39, July 2003.

[11] D. Moore, C. Shannon, and J. Brown. Code-Red: a case study on the spread and victims of an Internet worm. In *Proceedings of the ACM/USENIX Internet Measurement Workshop*, Marseille, France, November 2002.

[12] D. Moore, C. Shannon, G. Voelker, and S. Savage. Internet Quarantine: Requirements for Containing Self-Propagating Code. In *Proceedings of the 2003 IEEE Infocom Conference, San Francisco, CA*, April 2003.

[13] R. Permeh, M. Maiffret, and R. Permeh. eEye Digital Security Advisory .ida Code Red Worm. `http://www.eeye.com/html/Research/Advisories/AL20010717.html`, July 2001.

[14] RFC 3513: Internet Protocol Version 6 (IPv6) Addressing Architecture.

[15] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, San Jose, CA, USA, January 2002.

[16] S. Staniford, V. Paxson, and N. Weaver. How to 0wn the Internet in Your Spare Time. In *Proc. USENIX Security Symposium*, 2002.

[17] A. Wagner and B. Plattner. Peer-to-peer systems as attack platform for distributed denial-of-service. In *ACM SACT Workshop, Washington, DC, USA*, 2002.

[18] L. Wall, T. Christiansen, and R. L. Schwarz. *Programming Perl, 2nd Edition*. O'Reilly, 1996.

[19] N. C. Weaver. `http://www.cs.berkeley.edu/~nweaver/warhol.html`, 2001.

[20] C. C. Zou, W. Gong, and D. Towsley. Code Red Worm Propagation Modeling and Analysis. In *Proceedings of the 9th ACM conference on Computer and communications security, Washington, DC, USA*, November 2002.