Potential Strategies for High Speed Active Worms: A Worst Case Analysis

Nicholas Weaver
U.C. Berkeley BRASS group
<nweaver@cs.berkeley.edu>

Last Revised: March 24, 2002

Abstract

Active worms, malicious programs which spread in a completely autonomous manner, have the potential to rapidly spread across the internet. Two important questions which must be answered when constructing defenses is how fast a worm can spread and how long a given worm can remain a significant threat on the Internet, as these answers dictate requirements for defenses. There are multiple obvious strategies, such as hitlist scanning, topologically aware scanning, and local subnet scanning, which result in very fast worms, able to completely spread through the Internet in under an hour. Other strategies would greatly enhance a worm's staying power, including permutation scanning and an upgradeable design. By understanding these strategies, it is possible to specify requirements for defenses to try to prevent future outbreaks.

1 Introduction

Active worms, autonomous programs which spread through the network by searching, attacking, and infecting remote machines, have been a serious issue since the development of the Morris internet worm[11]. Such male-ware can carry arbitrarily malicious payloads which are spread rapidly to every vulnerable machine. In order to build defenses against future worm outbreaks, it is important to understand the capabilities of such worms: how fast they can spread and how long they can remain active on the Internet. The most effective potential worms needs to be considered, not what previous worms have demonstrated, in order to place requirements on the defenses needed to stop such worms.

The spread of active worms is generally limited by how quickly new potential hosts can be discovered. There are multiple strategies, including hitlist scanning (Section 3), topologically aware scanning (Section 4), and local subnet scanning (Section 5), for quickly identifying vulnerable hosts. A worm which uses some or all of these strategies combined with fast scanning routines would be very virulent, spreading worldwide in much less than an hour. These high speed worms can outpace any human-mediated defenses, requiring automatic defenses to prevent outbreaks.

The staying power of active worms is a function of rescanning strategies such as permutation scanning (Section 6), preventing the remote removal of the worm (Section 7, and an ability to exploit new attacks after the worm is released (Section 8). There are obvious solutions to all these problems which a worm author could employ.

Fortunately there are several published defenses (Section 9) which, if employed, could significantly reduce or eliminate such a worm's ability to spread. Additional defenses, including ISP coordination, are needed to seriously reduce a worm's staying power.

2 Previous Active Worms

An active worm, unlike the slightly more common mail worm, needs no human interaction to spread. It starts out on a single host and scans for other vulnerable machines on the Internet. When the scan finds a machine which the worm can potentially infect, it sends out a probe to infect the target. If successful, the worm transfers over a copy of itself to the new host, which begins running the worm.

There have been three highly significant active worm

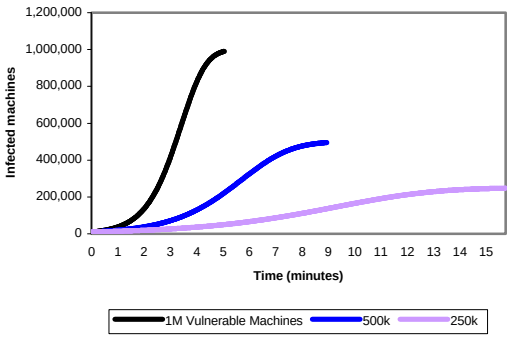

Figure 1: The effects of the fraction of vulnerable machines on worm propagation, using the simulator discussed later. All entries are for a hitlist (Section 3) of 10,000 machines, 100 scans/worm/second, permutation scanning (no halting). Graphs stop at 99% infection and do not include the time to process the hitlist.

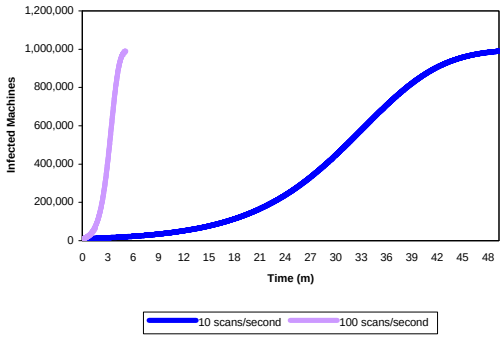

Figure 2: The effects of the scanning speed on worm propagation. All entries are for a hitlist of 10,000 machines, 1M vulnerable hosts, and permutation scanning (no halting). Graphs stop at 99% infection and do not include the time to process the hitlist.

outbreaks: the Morris Internet Worm, Code Red[6], and Nimda[2]. Additionally, many other less visible, slower spreading worms like Ramen[3] and Cheese[4] have been seen in the wild.

The main factor which limits how fast an active worm can spread is how fast new targets can be discovered, how many potential targets are available, and how fast they can be infected. As seen in Figure 1, fewer targets will cause a worm to slow down, because any given scan is less likely to find a valid target. The time it takes to infect a target slows a worm slightly, by limiting the rate at which new worms come online.

Worms have the potential to scan the net very quickly. Although a single scan may take anywhere from a few milliseconds (for a local host) to a second or longer (for a remote machine), a multithreaded worm can easily scan in parallel. A properly constructed scanner should be able to have many scans outstanding, even using a TCP-like back off strategy to insure that it is scanning at the maximum possible rate. 100 scans per second should be easily achievable, but even 10 scans per second results in a fast worm.

A worm is also likely to separate out the act of scanning and probing by only probing machines which the scan suggests are actually vulnerable. Code Red was indiscriminate in its probing, thus it tried to infect many non-vulnerable web servers. This has two negative effects from the worm's point of view: it plainly told the recipient machine that the source machine was running the worm (resulting in several anti Code Red web pages[1]) and it slowed down the rate of infection since such probes are a significant waste of effort.

Even very fast active worms only have minor effects on the modern network. Since the worm itself can be small (100k is a reasonable size, Code Red was only a few kilobytes), a probe to attempt an infection is smaller (1-2k or so), and the scan itself is miniscule (a few dozen bytes may be sufficient), the actual bandwidth requirements are surprisingly low. This is in marked contrast to mail worms, which have to send out copies of themselves in order to attempt to infect a host.

The only significant network effect is a marked increase in routing related requests, as the scanning worms keep trying to probe different machines throughout the world. This should have little effect on backbone routers, but Code Red did demonstrate effects on some routers near the periphery.

3 Hitlist Scanning

One of the biggest problems a worm faces is getting a significant initial population. Although a worm spreads exponentially during the early stages of infection, the time needed to infect the first 10,000 hosts dominates the infection time, as can be seen in Figure 3 and Figure 4.

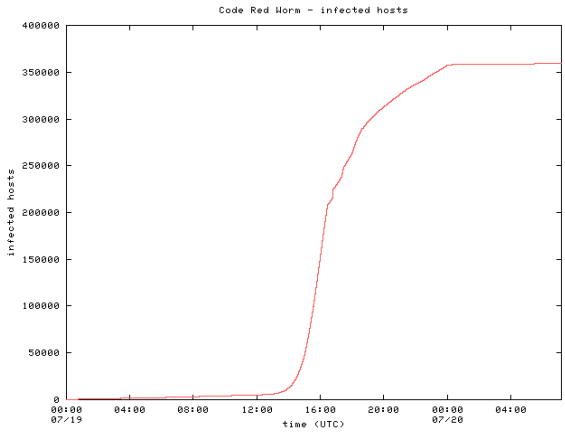

Figure 3: Code Red's Propagation Behavior, from David Moore's analysis[9]. Code Red v2 (a bug fix to use a random seed, the initial release was buggy.) seems to have been released at around 10:00 UTC.

There is a simple way for an active worm to overcome this obstacle: hitlist scanning. Long before the worm is released, the worm author collects a list of 10,000 to 50,000 potentially vulnerable machines with good network connections. The worm, when released onto an initial machine on this hitlist, begins scanning down the list. When it infects a machine, it divides the hitlist in half, communicating half to the recipient worm, keeping the other half.

This quick division insures that even if only 10-20% of the machines on the hitlist are actually vulnerable, an active worm will quickly go through the hitlist and establish itself on all vulnerable machines in under a minute. Although the hitlist may start at 200 kilobytes, it quickly shrinks to nothing during the partitioning. This provides a great benefit in constructing a fast worm by speeding the initial infection[1].

Constructing the hitlist is easy for a worm's author. Since the hitlist is constructed long before a worm is released, a slow scan would not be noticed. The Honeynet project[10] has shown that scans occur at alarming frequencies, thus another one could be conducted without people correlating it with the later worm release. Since such a scan is just to determine what OS and services a machine is running, not whether the targeted hole exists, it could be completed long before an exploit for the

[1]It is pointed out by Staniford et al[12] that if the hitlist comprises every vulnerable machine in the internet, complete infection can be achieved in under a minute

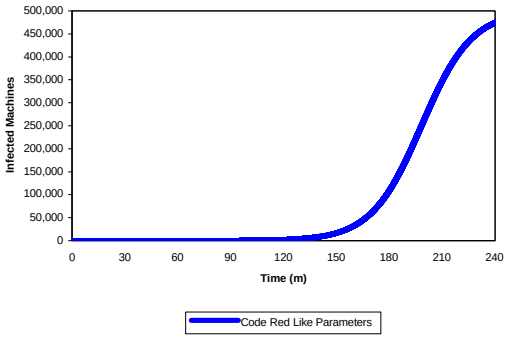

Figure 4: A simulated Code Red v2 like worm: 500,000 vulnerable machines, 10 scans per second, random scanning, starting on a single machine. Graph stops at 95% infection

worm is developed. Finally, public surveys such as the Netcraft Survey[13] could be used to generate the hitlist for some services without requiring a scan.

3.1 Performance Analysis of Hitlist Scanning

A small, abstract simulator of a worm's spread was built in the C language. This simulator assumes complete connectivity within a 2^{32} entry address space[2] using a pseudo-random permutation to map addresses to a subset of vulnerable machines. A 32 bit, 6 round variant of RC5 is used to generate all permutations.

Several parameters, including the number of vulnerable machines in the address space, the number of scans per second, the time to infect a machine, and the number infected during the hitlist phase, and the type of secondary scan (permutation, partitioned permutation, and random). The simulator assumes multithreaded scanning and infection routines which can have many simultaneous requests outstanding.

In order to roughly insure that the simulator is producing meaningful results, a rough simulation of Code Red v2 was performed, assuming 500,000 vulnerable hosts with a worm design capable of 10 scans/second[3]. The simulated worm takes approximatly 4 hours to reach gener-

[2]In general, the internet address space isn't completely connected, but it is close to fully connected. If a machine is not reachable from an arbitrary point on the external network, it is probably not reachable directly by a worm except through local scanning.

[3]Code Red v2 created 20 threads, each of which did an in-order random scan: create a connection, see if it is established, send the probe, wait for a response

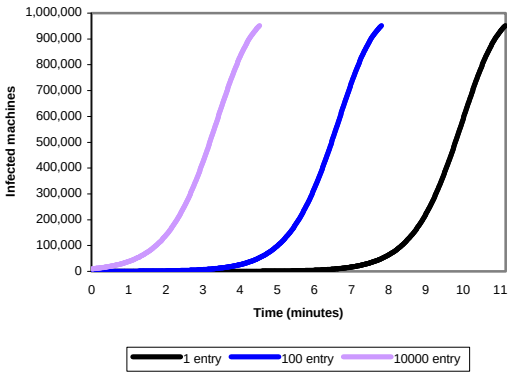

Figure 5: The effects of hitlist size on worm propagation. Permutation scanning, halting at 8 (Section 6) was used in all cases, 1,000,000 vulnerable hosts, 100 scans/second, 1 second to complete infection. Graphs stop at 95% infection and do not include the time to process the hitlist.

ally complete infection, only a factor of 2 off from the actual Code Red v2, which was the bug fix released at roughly 10:00 UTC and 18:00 UTC where most vulnerable machines were infected.

redo with 300,000 instead of 500,000.

Hitlist scanning has a huge effect on speeding a worm's propagation time. As can be seen in Figure 5, the speedup is immense. It takes almost three times as long for a worm with no hitlist to reach 95% infection, when compared to a 10,000 entry hitlist. Even for a slow scanning worm capable of only 10 scans/second, with 1 million vulnerable hosts and a hitlist containing 10,000 vulnerable hosts, it still would only require an hour to infect 95% of the net.

4 Topological Scanning

An alternative to hitlist scanning is topologically aware scanning, which uses information contained on the victim machine in order to select new targets. Email worms have used this tactic since their inception, as they harvest addresses from their victim in order to find new potential targets, while the Morris worm used this technique because of the very sparse address space when it was released.

Many future active worms could easily apply these techniques during the initial spread, before switching to a permutation scan once the known neighbors are exhausted. An active worm which attacked a flaw in a Peer to Peer application could easily get a list of peers from a victim and use those peers as the basis of its attack, which makes peer to peer applications (such as AOL Instant Messenger) highly attractive targets for worm authors. Although we have yet to see such a worm in the wild, these applications should be scrutinized for security.

Similarly, a worm attacking web servers could look for URLs on disk and use these URLs as seed targets. Since these are known to be valid web servers, this would greatly increase the initial spread by preferentially probing for likely targets.

Such topologically aware scanning can easily work to accelerate initial propagations, as the worm is initially able to find targets with a high degree of accuracy. The effects end up being very similar to hitlist scanning, making the initial growth occur much more quickly.

5 Local Subnet Scanning

One technique which Code Red II employed to great success is local subnet scanning. Instead of simply selecting machines at random, the worm preferentially scanned for targets on "local" addresses, those which were identical in the upper address range, a technique which Nimda copied. This allows a single copy of a worm running behind a firewall to rapidly spread to all other vulnerable machines which might otherwise be protected by the firewall.

There is nothing which prevents local subnet scanning from being used in conjunction with other scanning mechanisms or infection strategies: an initial fraction of the scans could be devoted to probing local machines, with the proportion eliminated once all local addresses are scanned. This results in worms which can quickly scan for potential targets when running behind a firewall without slowing the normal infection process.

6 Permutation Scanning

In a permutation scan, all worms share a common pseudo random permutation of the IP address space. Such a permutation can be efficiently generated using any block cipher of 32 bits with a preselected key: simply encrypt an index to get the corresponding address in the permutation, and decrypt an address to get its index.

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

This has the effect of providing a semi coordinated, comprehensive scan while maintaining the benefits of random probing. Each worm looks like it is conducting a random scan, but it attempts to minimize duplication of effort. This keeps the infection rate higher and guarantees an eventual comprehensive scan. After any particular copy of the worm sees several infected machines without discovering new vulnerable targets, the worm assumes that effectively complete infection has occured and stops the scanning process.

A timer could then induce the worms to wake up, change the permutation key to the next one in a prespecified sequence, and begin scanning through the new permutation, starting at its index and halting when another instance is discovered. This process insures that every address on the net would be efficiently rescanned at regular intervals, detecting any machines which came onto the net or was reinstalled but not patched, greatly increasing a worm's staying power within the net.

A further optimization is a partitioned permutation scan. In this scheme, the worm has a range of the permutation that it is initially responsible for. When it infects another machine, it reduces its range in half with the newly infected worm taking the other section. When the range gets below a certain level, it switches to simple permutation scanning and otherwise behaves like a permutation scan. It offers a slight but noticeable increase in scanning efficiency.

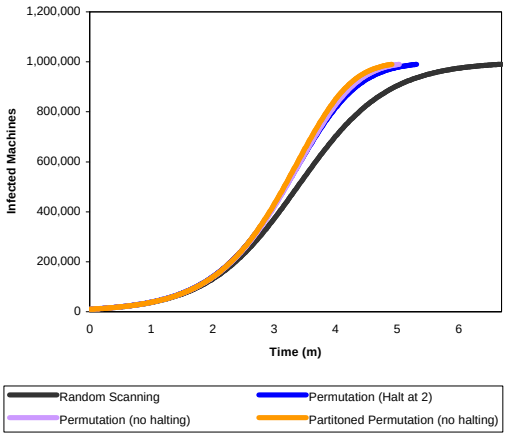

Figure 6: The effects of random, permutation (no halting), permutation (halt if 4 infections are seen without a new target), and partitioned permutation (no halting) scanning. All cases were for 1,000,000 vulnerable hosts, 10,000 entry hitlist, 100 scans/worm/second, 1 second to complete infection. Graphs stop at 99% infection and do not include the time to process the hitlist.

6.1 Performance Analysis of Permutation Scanning

Permutation scanning offers a significant gain towards the final end of a worm's spread, by insuring a comprehensive scan. Even with the halting approximation, Figure 6 shows the significant speedup achieved by permutation scanning. Once 95% infection is achieved, all the variants significantly surpass random scanning due to the greater efficiency.

Of further interest, in Figure 7, is the sensitivity to when a permutation scanning worm assumes that there are too few vulnerable machines and therefore stops scanning. Even for halting when 2 infected machines are found with no new target being discovered, the performance is still very good, which suggests that such an estimate is highly effective approximation of the scope of the infection.

Figure 8 shows how the number of worms still scanning has dropped down considerably as nearly complete infection is achieved, when halt at 2 is used. This means that, by the time near complete infection is achieved, a large fraction of the worms have gone silent, waiting to be woken up at a later time.

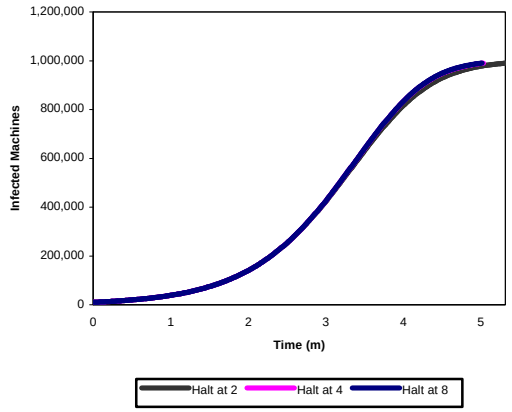

Figure 7: The effects of halt at 2, 4, and 8, on permutation scanning. All cases were for 1,000,000 vulnerable hosts, 10,000 entry hitlist, 100 scans/worm/second, 1 second to complete infection. Graphs stop at 99% infection and do not include the time to process the hitlist.

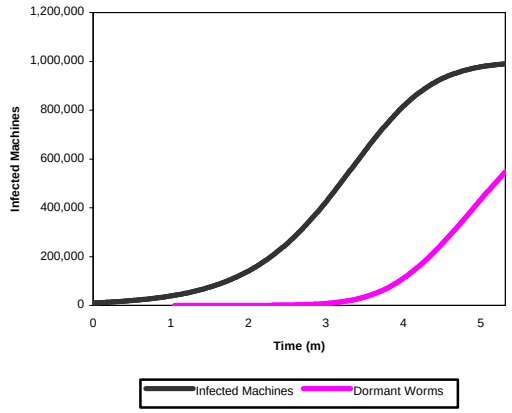

Figure 8: The number of dormant infections and total infections for a permutation scan which halts at 2. 1,000,000 vulnerable hosts, 10,000 entry hitlist, 100 scans/worm/second. Graph stops at 99% infection and do not include the time to process the hitlist.

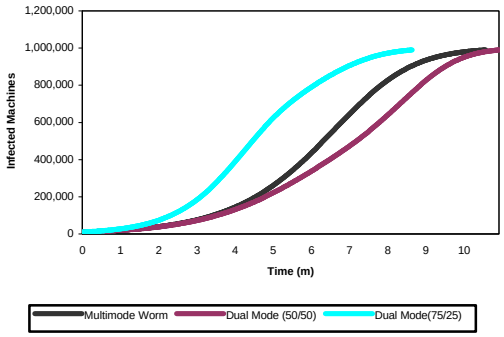

Figure 9: The number of infected machines for a multimode worm, a dualmode worm with equal distribution of holes (50/50), and a dual mode worm with unequal distribution of holes (75/25). For all cases, there are 1,000,000 vulnerable hosts, 10,000 entry hitlist (for the common exploit), permutation scanning (halt at 2.) The dual mode worms are 100 scans/worm/second, with the multimode worm being 50 scans/second (but scanning for both holes). Graph stops at 99% infection and do not include the time to process the hitlist.

6.2 Multimode and Dual Mode Worms

Both the Morris worm and Nimda were multimode worms: attempting to exploit multiple security holes in order to spread to more targets. An interesting question is how multimode worms interact with permutation scanning. One option is the classic multimode worm: when it scans a target, it searches for both security holes. The second is a dual mode worm: it starts out searching for the first hole, until it decides that the first hole has been completely exploited. Then it switches to exploiting the second hole.

Such a dual mode worm is easy to create. It starts out exploiting the first bug, but instead of halting during the permutation scan, it switches to the second bug. If a machine is infected through the second bug, it only scans for the second bug.

For an equal or nearly equal ratio, a multimode strategy is superior, while a dual mode strategy is more effective when there is an unequal ratio between the exploits. This phenomenon is easily seen in Figure 9, where the multimode worm outperforms the equal ratio dual-mode worm, but lags behind the unequal-ratio dual mode worm. This is due to the phenomenon where worms spread faster when more hosts are available.

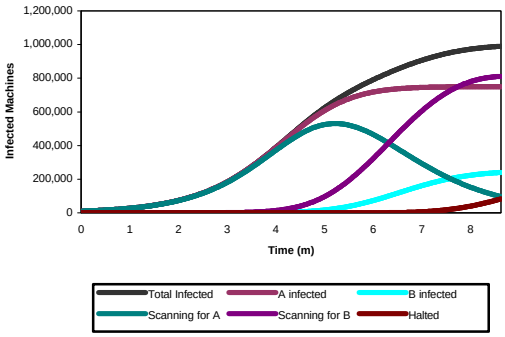

Figure 10: The total number of infected machines, infected by exploit A (75%) and by exploit B (25%) and the number of worms probing for exploit A, exploit B, and halted for a dual mode worm. 1,000,000 total vulnerable machines, 10,000 entry hitlist, 100 scans/worm/second, switch or halt at 2. Graph stops at 99% infection and does not include the time to process the hitlist

Since the unequal ration dual mode worm is first exploiting the more common bug, it can spread very quickly, with the comprehensive nature of permutation scanning insuring that the rarer bug is still exploited quickly.

The behavior can be further seen in Figure 10. As the first exploit's productivity is exhausted, the worms shift over to the second exploit. With such a large population scanning through the address space for the second exploit, even a relatively rare exploit can be quickly exhausted.

7 Countering Counterattacks

Although general antiworms, worms which remove other worms, are not a productive idea[4], responses to worm probes can be a highly effective defense against some worms. A good example are the many anti-Code-Red default.ida[1] pages which were created after Code Red II was released. These web servers would respond to a probe by Code Red II by counterattacking the infected web server, disabling the server, and resetting the machine: halting the code red infection and preventing reinfection. This is because a worm broadcasts its presence in order to spread, enabling active counter-attacks if the worm creates or exposes a security hole on its host.

[4]due effect of bugs in the antiworm

This suggests that any author who wants to increase the worm's staying power will simply not repeat the mistakes of Code Red II and other worms by simply closing the security holes used by the worm to infect a machine and not creating any generally open security holes. Such a worm would not be vulnerable to counterworms and antiworms. Therefore we can not assume that counter attacks and antiworms can form a viable defense against a sophisticated worm.

8 Distributed Control and Update Mechanisms

Some previous worms such as the Goner mail worm[5] contained primitive remote control code, allowing the authors and others to issue commands to a distributed DoS module through an IRC channel. Others have attempted to download updates from web pages. Both of these mechanisms, when employed, were quickly countered by removing the pages and tracking the channels. A more sophisticated method, direct worm to worm communication, has been discussed but not seen in the wild. Similarly, arbitrary updates have been discussed but not seen.

8.1 Distributed Control

In a distributed-control worm, each worm has a list of other known, running copies of the worm and an ability to create encrypted communication channels to spread information. Any new command has a unique identifier. Once a worm has a copy of the command, the command is first verified, spread to every other known worm, and then executed. This allows any issued command to be initially sent to an arbitrary worm, where it is then quickly spread to all running copies.

The key to such a network is the degree of connectivity maintained, in order to overcome worms being removed from the network and to hasten the spread of new commands. Although it is obvious that a worm could spread information to it's neighbors about other worms, creating a more connected, highly redundant network, it is useful to estimate the initial degree of connectivity without these additional steps.

If each worm node only knows about other worms it has probed, infected, or been probed by, the average connectivity is still very high. With 1M hosts, using permuta-

tion scanning (with no halting), the average degree of nodes in the worm network is 4 when 95% infection is achieved, and 5.5 when 99% infection is achieved. Additionally, each permutation based rescan will add 2 to the degree of every worm. Thus, after a couple of timer based rescans, the connectivity becomes very high.

Such a network could be used to quickly pass updates to all running copies, without having a single point of communication like that seen in previous worms. This increasing the staying power by preventing the communication channel from being coopted by others while allowing the author to control his creation with less risks of being tracked and traced.

8.2 Programatic Updates

Similarly, there is nothing which prevents a worm's commands from being arbitrary code. Many operating systems already support convenient dynamic code loading which could be readily employed by a worm's author or the worm could be written in a flexible language combined with an interpreter. By making the commands be general modules, a huge increase in staying power is achieved.

Of particular interest are new attack modules. If the author discovers a new security hole and creates a new attack module, this could be released into the worm network. Even if only a few thousand copies of the worm remain, this is enough of an installed base for a hitlist-style effect to occur upon introduction of a new attack module, quickly spreading the worm back through the net. Antivirus vendors will need heuristics to prevent reoutbreaks.

It is an interesting question whether it is possible for a worm author to release such a worm with the cryptographic modules correctly implemented. If the worm author attemts to build his own cryptographic implementation, this would be a significant weakness which could be exploited.

Yet there are many strong cryptographic applications and libraries which could be used by a worm author to provide the cryptographic framework, a good example being the OpenSSL[**?**] library which inclues SSL encryption (for link transfering), symmetric cyphers, hash functions, and public key cyphers and signatures to provide for code signing.

9 Developing Defenses

Preventative techniques which linguistically prevent buffer overflows, such as safe-C dialects[7], buffer overflow analysis[15], non-executable stack and heap policies, or Software Fault Isolation[16] are essential features of any defense: preventing the common holes from developing. Far too many exploitable buffer overflows have been reported, which a worm author could easily search for and exploit.

More importantly, defenses which contain an infection, such as fine grained access controls[8] or host based intrusion detection through program analysis[14], are an essential feature of a comprehensive defense. These are necessary because even with tools to catch the common faults, there still may be holes which a worm author could exploit. By creating robust secondary defenses, this greatly lessens the ability for someone to create a worm. Defense in depth offers significant advantages which should be exploited.

Yet there is onemore defense of note: a protocol for isolating infected and compromised machines from the net. Responding to infected machines is necessary because even a relatively small number of machines can cause severe disruptions through DDoS attacks or as launching pads for successive outbreaks. Even without an outbreak of such a damaging worm, this protocol is necessary because of the current prevalence of zombie machines and their effect on the internet. Developing such a protocol would be tricky, but possible.

10 Conclusions

Unfortunately, many obvious techniques such as these could result in significantly faster and longer lived internet worms. Although they have yet to be seen in practice, the techniques described here and elsewhere could easily be employed by worm authors. By analyzing these techniques, this suggests that developing and installing defenses should be a very high priority, as the current systems are highly vulnerable to fast moving worms.

The potential for fast worms is astonishing: optimized scanning routines combined with a reasonable hitlist size can produce worms which can easily spread worldwide in under an hour if a suitable hole is exploited. Similarly, permutation scanning combined with distributed updates

could create a worm which would remain a significant threat to weeks or months from the moment of initial release. The threat is too great to ignore.

11 Acknowledgments

This work is funded in part by support from DARPA, with additional support from the California MICRO program, ST Microelectronics, and Xilinx. Thanks to Michael Constant for helping develop the Hitlist scanning concept, Jon Kuroda, Jim Ausman, David Wagner, and Stuart Staniford for additional suggestions and comments.

References

[1] Das Bistro Project's anti-code-red default.ida. http://www.dasbistro.com/default.ida.

[2] CERT. Cert advisory ca-2001-26 nimdaworm, http://www.cert.org/advisories/ca-2001-26.html.

[3] CERT. Cert incident note in-2001-01, http://www.cert.org/incident_notes/in-2001-01.html.

[4] CERT. Cert incident note in-2001-05, http://www.cert.org/incident_notes/in-2001-05.html.

[5] CERT. Cert incident note in-2001-15, http://www.cert.org/incident_notes/in-2001-15.html.

[6] Cooperative Association for Internet Data Analysis. Caida analysis of code red, http://www.caida.org/analysis/security/code-red/.

[7] Trevor Jim, Greg Morrisett, Dan Grossman, Michael Hicks, James Cheney, and Yanling Wang. Cyclone: a safe dialect of c, http://www.research.att.com/projects/cyclone/papers/cyclone-safety.pdf.

[8] Peter Losccco and Stephen Smalley. Integrating flexible support for security policies into the linux operating system. In *Proceedings of the 2001 USENIX Annual Technical Conference*. USENIX, 2001.

[9] David Moore. The spread of the code red worm (crv2), http://www.caida.org/analysis/security/code-red/coderedv2_analysis.xml.

[10] The Honeynet Project. http://project.honeynet.org/.

[11] E. H. Spafford. The internet worm incident. In C. Ghezzi and J. A. McDermid, editors, *ESEC'89 2nd European Software Engineering Conference*, University of Warwick, Coventry, United Kingdom, 1989. Springer.

[12] Stuart Staniford, Gary Grim, and Roelof Jonkman. Flash worms: Thirty seconds to infect the internet, http://www.silicondefense.com/flash/.

[13] The Netcraft Survey. http://www.netcraft.com/survey/.

[14] David Wagner and Drew Dean. Intrusion detection via static analysis. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*. IEEE, 2001.

[15] David Wagner, Jeffrey Foster, Eric Brewer, and Alexander Aiken. A first step towards automatic detection of buffer overrun vulnerabilities. In *Proceedings of the Internet Society Network and Distributed Security Symposium*.

[16] Robert Wahbe, Steven Lucco, Thomas E. Anderson, and Susan L. Graham. Efficient software-based fault isolation. *ACM SIGOPS Operating Systems Review*, 27(5):203–216, December 1993.