Large Scale Malicious Code: A Research Agenda *†

Nicholas Weaver Vern Paxson Silicon Defense & ICSI Center for UC Berkeley Internet Research

son Stuart Staniford er for Silicon Defense

Robert Cunningham MIT Lincoln Laboratory

1 Executive Summary

The reliable operation of our networked computing infrastructure is essential to many governmental and corporate activities. Unfortunately, this infrastructure is highly vulnerable to automated attacks by computer worms: programs that propagate themselves to all vulnerable machines on the Internet. Such wide-scale malicious code is a major threat.

Previous worms, such as Code Red[25] and Nimda[12], were relatively minor: they contained no overtly malicious payload designed to affect the infected machine and attacked comparatively well-known vulnerabilities. Even so, they were moderately disruptive and highlighted the systemic vulnerabilities as the worms infected hundreds of thousands of machines in a few hours. Numerous companies and institutions lost a day of work while the computers were restored.

Future attacks can be considerably faster through some simple optimizations and alternate strategies, allowing all vulnerable machines to be infected in far less than an hour: faster than humans can react. Alternatively, some strategies don't accelerate the spread but make the attack much harder to detect[86].

An attacker using an otherwise unknown vulnerability could potentially corrupt millions of computers, if the vulnerable population is widespread. A malicious attacker could search or disrupt any information present on the infected machines, and/or use them to conduct wide-scale attacks on the Internet infrastructure. What makes the threat particularly serious is that the resources required to launch such an attack are comparatively small: a few skilled programmers and a small group of test machines.

There are several strategies possible, including *active scanning*, *topologically-aware*, *contagion*, *metaserver*, and *flash* attacks, which can't be detected or responded to by current systems. There are numerous possible payloads, such as data erasers, hardware-damaging routines, Internet-scale denial-of-service attacks, or widespread espionage, which could significantly affect the U.S. economy if contained in a widespread worm.

If our nation wishes to rely upon commodity networked computers for our day to day business, governmental, and military operations, we need to invest in several avenues of research to address the threat posed by the different families of malicious code. Much of this research must be government-sponsored because of the forward looking nature, the lack of a clear money-making proposition, and the requirement for widespread and proactive defenses.

This report aims to survey the different types of research necessary for addressing the threat, and, in particular, to then assess the priority of providing funding for the different areas. Some areas, while promising, are already being pursued by existing efforts or commercial entities; others are high risk, but with only modest promise; while still others have high promise and are currently undersupported. These latter form the highest funding priority, while the others should have less priority. (See Section 5 and subsequent sections for our specific funding priority recommendations.)

Much remains to be done to defend against worms. Although there is already considerable research in the area of creating more secure and robust computer systems, few

^{*}Distribution Statement "A": Approved for Public Release, Distribution Unlimited.

[†]The views, opinions, and/or findings contained in this article are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the Department of Defense.

of these features are easy to use or widely deployed.

Since there appear to be a limited number of strategies that enable a worm to find and infect new targets, it should be possible to create automated sensors which detect and respond to these various strategies. Once a worm is detected, it is then possible to institute reactions which throttle the worm based on its method(s) of propagation. Some type of automated response will be essential to slow the worm to the point where human reasoning again becomes relevant.

To succeed, improvements will be needed in tools that automatically perform an initial analysis of a worm based on its behavior: what it can infect, how it spreads, and particular features of its code. Such information can guide more precise responses and alert Internet sites if the worm poses a particularly significant threat.

Manual analysis is currently based on disassemblers, debuggers and similar tools, with current worms requiring extensive periods of time[79, 75]. Since even today's worms spread world-wide in less than half that time [25, 62, 86], the current manual analysis tools are too slow to aid in creating meaningful responses. By developing improved tools and other techniques, it should be possible to reduce analysis time considerably.

Significant effort is also needed in improving the response and recovery procedure. The current response relies on only loose coordination among individuals, with the few channels for updates being limited and susceptible to secondary attack. Considerable research is needed to develop recovery techniques which can automate this process and mechanisms which can resist a determined attack.

Cooperative defenses are essential for many facets of worm defense. Some may need to be mandated, while others may simply be subsidized. Cooperation offers numerous benefits. Many sensing and analysis schemes benefit from wide, cooperative deployments to increase sensitivity and robustness. Another benefit is derived from the global effects: some deployments can tolerate more significant, short term responses. Reactions that temporarily deny access to systems with a specific vulnerability will slow the overall spread of an infection.

We envision the possibility of a Cyber CDC to lead the development and deployment of these defenses (Section 4). There needs to be a considerable government role due to the common problems which worms present and the need for cooperative responses. Although any individual might only see a small risk to their own data, the overall risk is unacceptably high.

Contents

1	Executive Summary				
2	Wor	ms: Types, Attackers, and Enabling Fac-			
	tors		5		
	2.1	Families of Widespread Malicious Code .	5		
		2.1.1 Activation techniques	5		
		2.1.2 Propagation strategies	6		
		2.1.3 Propagation carriers	8		
	2.2	Toolkit Potential	8		
	2.3	Motivations and Attackers	8		
	2.4	Payloads	9		
	2.5	The Ecology of Worms	10		
	2.6	Potential Economic Damage	11		
3	Exis	ting Practices and Models	11		
	3.1	Cooperative Information Technology Or-			
		ganizations	11		
		3.1.1 U.SFunded Organizations	11		
		3.1.2 International Organizations	12		
		3.1.3 Commercial Entities	12		
	3.2	The Centers for Disease Control	13		
4	A C	yber CDC	14		
	4.1	Identifying outbreaks	15		
	4.2	Rapidly analyzing pathogens	15		
	4.3	Fighting infections	15		
	4.4	Anticipating new vectors	16		
	4.5	Proactively devising detectors	16		
	4.6	Resisting future threats	16		
	4.7	How open?	16		
5	Vulr	nerability Prevention Defenses	17		
	5.1	Programming Languages and Compilers .	17		
	5.2	Privilege Issues	19		
	5.3	Protocol Design	19		
	5.4	Network Provider Practices	20		
	5.5	Implementation Diversity	21		
	5.6	Synthetic Polycultures	21		
	5.7	Economic and Social	21		
6	Auto	omatic Detection of Malicious Code	21		
	6.1	Host-based Detectors	22		
	6.2	Network-level Detectors	23		
	6.3	Correlation of Results	23		

7 Automated Responses to Malicious Code	24
8 Aids to Manual Analysis of Malicious Code	26
9 Aids to Recovery	27
10 Coordination and Scaling	28
11 Policy Considerations	29
12 Validation and Challenge Problems 12.1 Establishing a Common Evaluation Framework 12.2 Milestones for Detection	
12.3 Milestones for Analysis	35
13 Conclusion	36
A Hardware devices	36
B Example Worm Potential Evaluation: Ha Life	lf 39

List of Tables

1	Possible milestones for detecting classes	
	of worms	32
2	A summary of the research areas in pre-	
	vention, ordered by grade and section	37
3	A summary of the research areas in au-	
	tomatic detection, ordered by grade and	
	section	37
4	A summary of the research areas in auto-	
	matic response, ordered by grade and sec-	
	tion	38
5	A summary of the research areas in man-	
	ual analysis and recovery, ordered by	
	grade and section	38
6	A summary of the research areas in vali-	
	dation, ordered by grade and section	38

2 Worms: Types, Attackers, and Enabling Factors

In order to understand the worm threat, it is necessary to understand the possible types of malicious code, the attackers who may employ them, and the potential payloads they could contain. Beyond that, it is necessary to understand why vulnerabilities arise and how they enable the spread of worms and other attacks.

2.1 Families of Widespread Malicious Code

A *worm* is a program that self-propagates across a network exploiting security flaws in widely-used services. They are not a new phenomenon, first gaining widespread notice in 1988[27].

We distinguish between worms and *viruses* in that the latter require some sort of user action to abet their propagation. As such, viruses tend to propagate more slowly. They also have more mature defenses due to the presence of a large anti-virus industry that actively seeks to identify and control their spread.

We note, however, that the line between worms and viruses is not all that sharp. In particular, the *contagion* worms discussed in [86] might be considered viruses by the definition we use here, though not of the traditional form, in that they do not *need* the user to activate them, but instead they exploit the user's activity in order to hide their spread. Thus, for ease of exposition, and for scoping our analysis, we will loosen our definition somewhat and term malicious code such as contagion, for which user action is not central to their activation, as a type of worm.

Related to these terminology distinctions, we begin our discussion of the different types of worms by first considering a worm's activation mechanism—the process by which a worm starts executing on a system—and a worm's propagation mechanism—the process by which a worm travels from one host to another.

2.1.1 Activation techniques

The means by which a worm is activated on a host can drastically affect how rapidly a worm can spread, because some worms can arrange to be activated nearly immediately, whereas others may wait weeks or months to be activated.

Human Activation The slowest activation approach requires a worm to convince a local user to execute the local copy of the worm. Since most people do not want to have a worm executing on their system, these worms rely on a variety of social engineering techniques. Some worms such as the Melissa virus[9] indicate urgency on the part of someone you know ("Attached is an important message for you"); others, such as the Iloveyou[10] attack, appeal to individuals' vanity ("Open this message to see who loves you"); and others, such as the Benjamin[88] worm appeal to greed ("Download this file to get copyrighted material for free"). Although Melissa was a word macro virus-a piece of code written in Microsoft Word's built-in scripting language embedded in a Word document-later human-initiated worms have usually been executable files which, when run, infect the target machine. Furthermore, while some worms required that a user start running a program, other worms exploited bugs in the software that brought data onto the local system, so that simply viewing the data would start the program running (e.g., Klez[29]). The continued spread of these worms is disturbing can be effectively used as secondary vectors¹[12] and or to install additional malicious software such as programs which allow an attacker to control a machine[28].

These attacks can be resisted by designing transport program user interfaces to disallow direct execution of programs. Virus spread can also be slowed by implementing virus filters in transport servers. If there is a strong need to allow software to perform tasks, then those tasks should be limited by a sandbox to a few enumerated possibilities. Ideally, unknown executables would be quarantined, even if there is no signature match, so that previously unknown worms are halted.

Scheduled Process Activation The next fastest worms activate using scheduled system processes. Such programs can propagate through mirror sites (e.g., OpenSSH Trojan[48]), or directly to desktop machines. Many desktop operating systems and applications include auto-updater programs that periodically download, install and run software updates. Early versions of these systems did not employ authentication, so an attacker needed only to serve a file to the desktop system[32]. Other systems pe-

¹A secondary spreading mechanism can often benefit a worm by enabling more targets to be attacked or as a device to cross defenses such as firewalls.

riodically run backup and other network software that includes vulnerabilities. The skills an attacker requires to exploit these depends on the scheduled process's design and implementation: if the attacked tool does not include authentication, a DNS redirection attack will suffice, but if it does, then the attacker might need to acquire the private keys for both the update server and code signing.

Self Activation The worms that are fastest activated are able to initiate their own execution by exploiting vulnerabilities in services that are always on and available (e.g., Code Red[25] exploiting IIS Web servers) or in the libraries that the services use (e.g., XDR[13]). Such worms either attach themselves to running services or execute other commands using the permissions associated with the attacked service. Execution occurs as soon as the worm can locate a copy of the vulnerable service and transmit the exploit code. Currently, preventing these attacks relies on running software that is not vulnerable, although the effect of an attack can be reduced by limiting the access of services that are always on.

2.1.2 Propagation strategies

As mentioned above, in order for a malicious program to run on a victim machine, it must somehow have its code introduced onto the victim. This code can be machine code, or it can be high level commands to an existing program. In order to propagate, malicious programs need to discover new victims and to distribute their code to the victims.

The distribution of code can either be one-to-many, as when single site provides a worm to other sites, many-tomany, as when multiple copies propagate the malicious code, or a hybrid approach. In general, many-to-many distribution can be considerably faster, if a limiting factor is the time it takes to perform the distribution. Many-tomany distribution also removes the ability for others to block further distribution by removing the source of the malicious code from the Internet.

There are a number of techniques by which a worm can discover new machines to exploit: scanning, external target lists, pre-generated target lists, internal target lists, and passive monitoring. Worms can also use a combination of these strategies.

Scanning: Scanning entails probing a set of addresses to identify vulnerable hosts. Two simple forms of scan-

ning are *sequential* (working through an address block from beginning to end) and *random* (trying addresses out of a block in a pseudo-random fashion). Due to its simplicity, it is a very common propagation strategy, and has been used both in fully autonomous worms[25, 12] and worms which require timer or user based activation[54]. Scanning worms spread comparatively slowly compared with a number of other spreading techniques, but when coupled with automatic activation, they can still spread very quickly in absolute terms.

There are currently few defenses in place to respond to scanning worms. The previous worms in this class have only exploited known and largely patched security holes or very small populations of machines[60], and therefore infected relatively few machines. Code Red I compromised about 360,000 machines[62], a small fraction of the estimated 10,000,000 machines running IIS[65], though the evidence indicates this may have been essentially the entire publicly-accessible population of IIS machines[86].

More sophisticated scanning techniques that incorporate bandwidth-limited routines², a preference for local addresses, and/or permutation of the search order[86] offer substantial improvements in performance. Other application-specific optimizations relying on different target-selection techniques offer even greater speedups (see below).

Except for the exploit, scanning worms are not application-specific. Thus an attacker can add a new exploit to an existing worm. The Slapper[81] worm was one such case, where the attacker inserted a new exploit into the Scalper[60] source code. This suggests that the window between when a vulnerability is released and when a worm appears will shrink to nearly zero, as the general scanning worm framework can be expressed as a toolkit.

In general, the speed of scanning worms is limited by a combination of factors, including the density of vulnerable machines, the design of the scanner, and the ability of edge routers to handle a potentially significant increase in new, diverse communication.

For these worms, the worm's spread rate is proportional to the size of the vulnerable population. Code Red I required roughly 12 hours to reach endemic levels, but

²Many worms, such as Code Red, used scanning routines which are limited by the latency of connection requests rather than the throughput by which requests can be sent

could have easily taken only 2 hours if it contained sophisticated scanning routines or targeted a more widespread vulnerability[86].

On the other hand, scanning is highly anomalous behavior, so it should be possible to effectively detect scanning worms as being very different from normal traffic. Similarly, if most hosts are limited in the amount of scanning they can perform, this greatly slows the speed of a worm.

Pre-generated Target Lists: An attacker could obtain a target list in advance, creating a "hit-list" of probable victims[86]. A small hit-list could be used to accelerate a scanning worm, while a complete hit-list creates a "flash" worm, capable of infecting all targets *extremely* rapidly.

The biggest obstacle is the effort to create the hit-list itself. For a small target list, public sources are readily available. Comprehensive lists require more effort: either a distributed scan or the compromise of a complete database. Like scanning worms, most of the code is application independent, suggesting that flash worms can also use toolkits in their implementation.

Externally Generated Target Lists: An external target list is one which is maintained by a separate server, such as a matchmaking service's *metaserver*. (A metaserver keeps a list of all the servers which are currently active. For example, in Gamespy[39] maintains a list of servers for several different games.) A metaserver worm first queries the metaserver in order to determine new targets. Such a worm could quickly spread through a game like HalfLife[85] or others. This technique could also be used to speed a worm attacking web servers, for example by using Google as a metaserver in order to find other web servers to attacks.

We have not seen a metaserver worm in the wild, but the potential is significant due to the great speed such a worm could achieve. On the other hand, the process of querying the metaserver is application specific, reducing the ability of toolkits to reduce the worm author's workload.

Internal Target Lists: Many applications contain information about other vulnerable machines on every host running the service. Such target lists can be used to create *topological* worms, where the worm searches for local information to find new victims by trying to discover the local communication topology.

The original Morris worm[27] used topological tech-

niques including the Network Yellow Pages, /etc/hosts, and other sources to find new victims. (Since the Internet at the time was very sparse, scanning techniques would have been ineffective.)

Topological worms can potentially be very fast. If the vulnerable machines are represented as vertices in a directed graph $G = \{V, E\}$, with edges representing information about other machines, the time it takes for a worm to infect the entire graph is a function of the shortest paths from the initial point of infection. For applications that are fairly highly connected, such worms are incredibly fast.

Although topological worms may present a global anomaly, the local traffic may appear normal. Each infected machine only needs to contact a few other machines. Since these are known machines, these may even represent normal destinations. This suggests that highly distributed sensors may be needed to detect topological worms.

Fortunately, extracting the topological information is often highly application-specific, which reduces the ease of constructing toolkits. An exception is for email worms, where there have already been toolkits [80] providing common mechanisms.

Passive: A passive worm does not seek out victim machines. Instead, they either wait for potential victims to contact the worm or rely on user behavior to discover new targets. Although potentially slow, passive worms produce no anomalous traffic patterns which potentially makes them highly stealthy. *Contagion*[86] worms are passive worms which rely on normal communication to discover new victims..

There have been many passive worms, such as the Gnuman[46] bait worm and the CRClean[50] "antiworm" (see Section 9 for more discussion of anti-worms). Gnuman operates by acting as a Gnutella node which replies to all queries with copies of itself. If this copy is run, the Gnuman worm starts on the victim and repeats this process. Since it requires user activation and is comparatively simple, it spreads very slowly.

CRClean took did not require human activation. This worm waits for a Code Red related probe. When it detects an infection attempt, it responds by launching a counterattack. If this counterattack is successful, it removes Code Red and installs itself on the machine. Thus CRClean spreads without any scanning.

2.1.3 Propagation carriers

The means by which propagation occurs can also affect the speed and stealth of a worm. A worm can either actively spread itself from machine to machine, or it can be carried along as part of normal communication.

Self-Carried: A self-carried worm actively transmits itself as part of the infection process. This mechanism is commonly employed in self-activating scanning or topological worms, as the act of transmitting the worm is part of the infection process. Some passive worms, such as CRClean[50], also use self-carried propagation.

Embedded: An embedded worm sends itself along as part of a normal communication channel, either appending to or replacing normal messages. As a result, the propagation does not appear as anomalous when viewed as a pattern of communication. The contagion strategy[86] is an example of a passive worm that uses embedded propagation.

An embedded strategy, although stealthy, only makes sense when the target selection strategy is also stealthy. (Otherwise, the worm will give itself away by its target selection traffic, and reaps little benefit from the stealth that embedded propagation provides.) Thus a scanning worm is unlikely to use an embedded distribution strategy, while passive worms can benefit considerably by ensuring that distribution is as stealthy as target selection.

The speed at which embedded worms spread is highly dependent on how the application is used, and represents a significant, application-dependent unknown. A related question is how far from the natural patterns of communication such a worm could deviate in order to hasten its propagation without compromising its stealthiness.

2.2 Toolkit Potential

As noted in the previous section, some target selection strategies lend themselves well to the creation of *toolkits*: large reusable structures where a small amount of additional code can be added to create a worm. Early versions of both application-independent[80] and application-dependent [80, 91] toolkits have been seen in the wild, and it is likely that such toolkits will become more widespread and sophisticated. There is nothing inherent in worm development that would limit the potential for developing such a toolkit. The application independent portions of a toolkit will contain code for scanning (both naive and sophisticated approaches) and transporting payloads. Other code will help with obfuscation or encryption to resist signature analysis. Finally, code that damages a system can also be independently developed and tested on a single, locally controlled host. Since these subsystems can be designed, developed and tested independent of exploits, attackers can complete these components in advance of assembling a worm. Indeed, it is possible that one of the already released but impotent worms was a test of the distribution portions of such a system.

Since the only work needed to release toolkit-based worms is integrating the exploit, the time between vulnerability information and worm release will quickly shrink to nearly zero, and the skill required to create such worms will also shrink.³

2.3 Motivations and Attackers

Although it is important to understand the technology of worms, in order to understand the nature of the threat, it is also important to understand the motivations of those that launch the attacks, and to identify (where possible) who the attackers are. This is a representative list organized by motivation; it is not an exhaustive enumeration.

Pride and Power: Some attackers are motivated by a desire to acquire (limited) power, and to show off their knowledge and ability to inflict harm on others[74]. The people who do this are typically unorganized individuals who target randomly; if they discover a system that is vulnerable to an attack they possess, then they are likely to execute the attack.

Commercial Advantage: Since the U.S. economy has grown heavily dependent on computers for day to day operation, a major electronic attack targeted against a single domain could seriously disrupt many companies that rely on Internet-based transactions. Such disruption could be used by an attacker wishing to profit by manipulating financial markets via a synthetic economic disaster, or by competitors that wish to limit buyers' access to a seller's

³The recent Scalper[60] worm was released only 10 days after the exploit was published and the source code is freely available. Slapper[81] reused the Scalper code base, effectively using the scalper source as a toolkit.

wares. International companies or organized crime members may participate in this type of attack, and the targets range from specific companies to economic infrastructure.

Extortion: Another potential profit motive is extortion. Since a well-constructed worm could launch an unstoppable DOS attack, major e-commerce or portal companies could be threatened unless payment is arranged. Such a worm could be launched by individuals or organized groups.

Random Protest: A disturbed person (such as the "Unabomber," Theodore Kaczynski) who wishes to disrupt networks and infrastructure and who has studied Internet systems and security could readily create a worm. The release of a truly destructive, optimized worm requires a level of patience and meticulousness not commonly seen, but definitely present in individuals like Kaczynski. Such individuals may search for a "zero-day exploit" (one unknown to the public community) in a common application, and would probably be more likely to construct a topological worm or similar attack which already requires application-specific programming.

Political Protest: Some groups wish to use the Internet to publicize a specific message and to prevent others from publicizing theirs. Individuals or organizations with local, national, and international presence can be involved. Targets include organizations with competing goals, or media outlets that are perceived as critical of an organization's goals. As one example, the Yaha Mail worm[58] was written as a tool of political protest by unknown parties claiming affiliation with Indian causes, to launch a DOS attack on Pakistani governmental web sites.

Terrorism: Terrorist groups could employ worms to meet some of their objectives. Since Internet-connected computers are a First World development, and major multinational concerns rely heavily on desktop machines for day to day operation, payloads could be selective to only execute in large, networked environments, making worms highly attractive weapons for those who believe that large corporations are an evil, as well as those with animosity directed against particular nations or governments. Attackers could include Al-Quaeda[17] or splinter groups derived from the antiglobalization movement, or groups such as ELF[37] or ALF[36], which claim to engage only in economic terrorism.

Cyber Warfare: As the U.S. is heavily dependent on computing infrastructure for both economic and govern-

mental needs, other nations with a significant interest in U.S. economic disruption could plausibly launch an electronic attack, either as a preemptive strike, or in response to U.S. action, or in conjunction with a physical strike. Along with large e-commerce sites, critical infrastructure, networked military, and governmental computers would be primary targets for such worms. Such attacks would be particularly appealing to nations without well-developed Internet infrastructure, as they would stand little to lose in terms of the worm attacking their hosts, too, or from a possible cyber counter-attack. The potential anonymity of cyber attacks also makes its use attractive for "cold war" situations, and for possibly framing others as the apparent perpetrators.

2.4 Payloads

Different sorts of attackers will desire different payloads to directly further their ends. Most of the following types of payloads have been seen in the wild.

None/nonfunctional: By far the most common is simply a nonexistent or nonfunctional payload. A worm with a bug in the propagation code usually fails to spread, while bugs in the payload still leave the worm able to spread. Such a worm can still have a significant effect, both through traffic[27] and by actively advertising vulnerable machines.

Opening Backdoors: Code Red II opened a trivialto-use privileged backdoor on victim machines, giving anyone with a web browser the ability to execute arbitrary code. This even gave rise to anti-Code-Red sites[69] which exploited the backdoor with the commands to disable IIS and reset the machine.

Remote DOS: Another common payload is a Denial of Service (DOS) attack. Code Red, Goner, and Yaha have all contained DOS tools, either targeted at specific sites or retargetable under user control. Distributed DOS (DDOS) tools such as Stacheldraht[21] have included stealthy and encrypted communication channels.

We have yet to see an attacker take advantage of Internet-scale DOS opportunities. With 100,000 or 1,000,000 controllable "zombies", the attacker could target the DNS system, update sites, response channels, possibly all at the same time.

Receive Updates: Past worms such as W32/sonic[89] have included a crude update mechanism: querying web

sites for new code. W32/hybris[28] also checked Usenet newsgroups and cryptographically verified the modules before execution. Similarly, DDOS tools have also enabled updates to the zombie program[22]. A controllable and updateable worm could take advantage of new exploit modules to increase its spread, enable sophisticated additions to the worm's functionality after release, and fix bugs after release.

Espionage: SirCam[11] performed inadvertent espionage, by attaching random files to its mailings, but a worm could just as easily preferentially search for document with various keywords, credit card numbers, or similar information. A worm could also "wardial" any modem⁴ to conduct further reconnaissance for later, non-Internet based attacks.

Data Harvesting: Criminals are sometimes interested in identify theft, and significant subsets of the blackhat community are involved in harvesting credit cards[7] and could use worms to search for this information. After discovery, the results could be encrypted and transmitted through various channels.

Data Damage: There have been many viruses and email worms, such as Chernobyl[53] or Klez[29], which contained time-delayed data erasers. Since worms can propagate much faster, they could start erasing or manipulating data beginning at the moment of infection.

Hardware Damage: Although the diversity of BIOSs prevents a general reflashing, it would be possible for a worm to include reflashing routines for one or two of the most common BIOSs, using the same mechanisms employed by the Chernobyl virus[53]. Since the FLASH ROMs are often soldered to the motherboard, such an attack could effectively destroy particular motherboards.

Coercion: A coercive payload does no damage unless the worm is disturbed. Such a worm attempts to remain entrenched by giving the user a choice: allow the worm and suffer no local damage, or attempt to eliminate the worm and risk catastrophic results.

2.5 The Ecology of Worms

For all the sophisticated strategies and potential payloads, worms can only exist if there are security or policy flaws they can exploit. Thus it is important to understand why such vulnerabilities exist and how they enable worms to operate. We refer to this surrounding context as the "ecology" of worms.

It may be tempting to say that we could build secure systems which will not have exploitable vulnerabilities. However, even highly secure software systems with reputations for robustness and which have received considerable security scrutiny including multiple code reviews, such as OpenSSH, OpenSSL and Apache, have contained major security holes. Products from other vendors, including Microsoft, are notorious for the volume of patches and security issues. It is critical to understand why vulnerabilities continue to exist.

Application Design: A significant factor in prevalence of vulnerabilities is the structure of the application and protocols. Some design features can make a system either considerably more or considerably less vulnerable to worm activity, including the pattern of communication, pattern of reachability, the maturity and quality of the code, the breadth of the distribution, and the selection of programming language.

It is desirable for a third party, such as a Cyber CDC, to perform audits of widespread applications to determine vulnerabilities and resistance to worm based attacks. A sample of what such an examination may look like is included in Appendix B.

Buffer Overflows: One of the largest sources of vulnerabilities is the continued use of the C and C++ languages, which allows buffer overflow attacks. These attacks represent roughly 50% of the major security flaws over the past 20 years. Most other programming languages are immune to such problems, and several technologies have been developed which can mitigate or prevent some or all of these attacks, such as StackGuard[19], Software Fault Isolation[96],⁵ unexecutable stacks and heaps, and "Safe C" dialects like CCured[64] and Cyclone[49]. Yet none of these have been widely adopted. See Section 5.1 for detailed discussion.

Privileges: Mail worms and potentially other types of worms often rely on the observation that programs are granted the full privileges of the user who operates them. This lack of containment is commonly exploited by mali-

⁴Wardialing is the process of scanning for telephone numbers which are attached to answering modems or similar devices.

⁵Which was being commercialized by Colusa software[84] before its purchase by Microsoft.

cious code authors.

Application Deployment: Widespread applications are more tempting targets for worm authors, especially those who would search for unknown exploits. Although even rare applications may have worms[60], widespread applications are of particular interest because of the increased speed of infection and the greater number of potential targets.

Economic Factors: Making programs robust and debugged represents a significant fraction of their development cost. Thus, unless the number of bugs and vulnerabilities is beyond customer tolerance, there are significant economic incentives to release buggy code.

Monocultures: Finally, there is the tendency for computing systems to form monocultures, which are inherently vulnerable to fast moving pathogens. Monocultures arise from various sources, including ease of administration, commonly taught and understood skills, and monopolistic behaviors.

2.6 Potential Economic Damage

It is difficult to estimate the potential damage to the Internet as a function of a worm outbreak. Previous damage figures, such as the widely reported \$2 billion cost for Code Red[24] and its variants, are often controversial, since many of the costs are either transitory disruptions (which cause little real damage) or represent questionable cases (does one consider the cost of post-outbreak patching as a a worm-associated cost, but pre-outbreak patching an unrelated cost?).

Another concern is simply that previous worms have not represented significantly malevolent attacks: Code Red infected approximately 3% of the IIS installations on the Internet and did not carry an overtly damaging payload. A worm which attacks a more widespread vulnerability in a common service could plausible represent over a hundred billion dollars in direct damage and with difficult-to-estimate but large additional indirect damages—would cause serious harm to the U.S. economy.⁶

3 Existing Practices and Models

This report does not attempt to cover all institutions involved in computer security or malicious code response, but instead provide a general overview of the various entities. Although there are many institutions in place, none are prepared or developing significant responses to the threat of computer worms. Since worms can propagate much faster than other threats, the existing infrastructure and institutions are generally not directly applicable, because existing institutions are centered around human rather than automated—detection, analysis and response, They are also generally limited in scope, and are centered around reactive models.

Although previous worms have been comparatively slow, they still spread faster than responses could be generated. This implies that institutions which seek to address this threat need to invest in automated detectors and response mechanisms. Yet there is no indication that the existing institutions are engaged in active development of strategic responses. Furthermore, there is a lack of direct economic incentive: worms are a global threat to the Internet, best resisted when everyone capable of mounting a defense does so. For many forms of global anti-worm defense, individuals reap little marginal benefit from deploying themselves, and hence lack economic incentives to do so.

3.1 Cooperative Information Technology Organizations

3.1.1 U.S.-Funded Organizations

CERT/CC: The Computer Emergency Response Team Coordination Center (CERT/CC) (www.cert.org) is a center located at the Software Engineering Institute, a federally funded research and development center operated by Carnegie Mellon University. The institution was founded in 1988, two weeks after the Morris worm[27] was released, to aggregate and present information about security vulnerabilities to system and network administrators, technology managers, and policy makers. Although CERT/CC acquires information from anyone connected to the Internet, the organization describes its field of vision as being centered on the needs of the U.S. Department of Defense. CERT/CC teams are divided into vul-

⁶For obvious reasons, details of the worst-case analysis where these damage figures are derived is currently not part of the public report.

nerability handling (analyzing flaws in Internet systems), incident handling (measuring exploitation of flaws) and artifact analysis (studying intruder-developed code). All of these teams rely on reports and data provided by external sites and mailing lists. CERT/CC's notes, reports and databases rely on human analysis and aggregation, and provide a model for human-readable descriptions of vulnerabilities and incidents. To address automated attacks, CERT/CC would require new tools and procedures.

IAIP: The Information Analysis and Infrastructure Protection Directorate was founded as a portion of the Department of Homeland Security (www.dhs.gov) in 2003 by unifying the protection activities of the Critical Infrastructure Assurance Office in the Department of Commerce and the National Infrastructure Protection Center of the FBI with the response functions of the Federal Computer Incident Response Center of the General Services Administration. A key task performed by the new organization is to provide Indications and Warning Advisories for physical and cyber events. The agency will also coordinate emergency preparedness for the telecommunications sector. The IAIP (as currently envisioned) will probably not develop tools for rapid worm analysis and detection, as it is focused on slower, human-time analysis and criminal prosecution. Nevertheless, tools built for a Cyber-CDC should also support forensic analysis so that they could be used by those at IAIP.

ISACs: Information Sharing and Analysis Centers, established by organizations now part of the Department of Homeland Security (http://www.dhs.gov/dhspublic/display?theme=73),

are intended to share information and work together to better protect the economy. The ISACs are responsible for defining "best security practices" and for collecting relevant security background data. This data is used to define normal and elevated levels of threats. Perhaps the most closely related ISAC is the shared Information Technology ISAC (https://www.it-isac.org/), which identifies an alert condition level or AlertCon that indicates the current relative IT-related threat level. The organization shares information provided by commercial security firms and members, but does not appear to have plans to develop its own tool set.

3.1.2 International Organizations

FIRST: The Forum of Incident Response and Security Teams (FIRST) (www.first.org) was formed in 1995, to enable international teams to work together. Each team identifies contact addresses and phone numbers and a constituency that they represent, so other teams can quickly communicate with response teams. FIRST has an annual conference that includes tutorials and technical sessions on security related topics. The organization could be used as an information distribution mechanism, but lacks a centralized site or tools to perform automated analysis.

Public Mailing Lists: Many mailing lists, such as **Bugtraq**[33], serve as general discussion forums for security issues, including warnings about new vulnerabilities and exploits, analysis of incidents and attacks, and notification of newly discovered worms. These mailing lists represent a significant portion of our defenses against attacks. They are excellent resources for recovery, but offer little warning in the case of virulent attack, when comparing the potential speed of a worm with the response of the mailing lists.

3.1.3 Commercial Entities

Anti-virus Companies: The large computer anti-virus industry has a primary mission of protecting customers from malicious code attacks. The first computer virus on a PC was discovered in 1986, and by 1990 several U.S. and European companies were offering anti-virus software to address the problem (e.g., McAfee[3], Norton[87] and Norman[66]). The industry has several formal and informal ways to share information that are relevant to a Cyber-CDC, but lacks the tools and the profit motive to develop rapid-analysis tools.

Information sharing is done via several organizations. Industry members and researchers share information, primarily in the form of collected viruses, via the Computer Anti-Virus Researchers Organization (CARO). When members receive a virus (e-mailed to them by their customers or by virus writers themselves) the virus is sent to other organizations via encrypted mail. Each organization then independently analyzes the virus, develops detection mechanisms specific to their systems, and usually develops a description of the virus for their web site.

The European Institute for Computer Anti-Virus

Research[26] has a broader membership that includes universities, and has established a code of ethics that all members must agree to prior to joining. Some system administrators (with help from vendors) share information with each other about emerging threats via AVIEN's early warning system[4]. Finally, most companies have a web site that identifies and explains specific viruses.

The industry toolset has a few limitations. The greatest is that most anti-virus tools are aimed at protecting attacks against files; only a few systems protect against worms that remain in memory, and this protection is only in its infancy. Information sharing is limited to sharing viruses themselves due to the competitive nature of the industry. Tools used are the more common set of de-compilers and disassemblers, and complex worms can require weeks to understand.

Network based Intrusion Detection System Vendors: There are many companies now selling network intrusion detection systems designed to alert and respond to network threats. These systems are not designed to respond to worm-like attacks, but are mostly focused around responding to known attacks. What is of greater concern is that worms probably require coordinated responses not present in the current deployed infrastructure.

Centralized Security Monitoring: Security Focus's DeepSight Threat Management Service[31] aggregates alerts from participating organizations who agree to share automated intrusion detection and other information. This information is then aggregated to provide incident analysis and other information for subscribers. The biggest limitations result from the subscriber model: only those who are willing to contribute receive any benefits.

A similar service—Managed Security Monitoring is offered by companies like Counterpane Internet Security[18]. Counterpane deploys sensors at customer locations, where they are centrally monitored by automatic and manual tools. Again, the largest limitation is the threat model and the subscriber model: worms are currently not perceived as a significant threat, thus there is little economic incentives to deploy defenses. Similarly, as a relatively expensive commercial service, such services are not widely employed.

Training Organizations: Numerous companies offer security training for practitioners. Among these are courses offered by vendor-neutral companies (e.g., the System Administration, Networking and Security or SANS institute, www.sans.org, and Foundstone, www.foundstone.com); by operating system companies (e.g., Red Hat Linux[78] and Microsoft[45]); and those by application system companies (e.g., IBM's Tivoli[44]). None of these organizations develop tools for analyzing fast-moving worms, although all of the courses apply to improving local security.

Limited Scope of Commercial Response: We finish with the observation that industry is unlikely to develop sophisticated detection, analysis and response tools, because (1) complex worms have historically only appeared a few times per year, and the worms have not inflicted significant damage (to the host site), and (2) there is no clear way to generate additional revenue with such tools. The situation is analogous to a "tragedy of the commons," with the Internet ecology as the commons. Thus, to get out in front of the worm threat will require a government-level response; the private sector by itself is not suited to the scale of the problem. We return to this point in Section 4.

3.2 The Centers for Disease Control

What is now known as the Centers for Disease Control and Prevention (www.cdc.gov) was formed as a single entity in 1946 to combat Malaria, which was then endemic in the southern U.S.[8]. However, its cultural roots are intertwined with the evolution of the Public Health Service and go back to the earliest days of the Republic. To address the challenges of new diseases brought to the U.S. by immigrant populations, Congress had by 1921 passed a sequence of laws and reorganizations placing all state quarantine stations under federal control.

Today the CDC provides specific disease-oriented research resources, surveillance of the health status of the U.S. and direct assistance to local health authorities in combating outbreaks that strain local resources, including bio-terrorism.

Surveillance depends on both data collection and extensive databases of normal background. In general, individual caregivers do not communicate directly with the CDC but rather with their local public health authorities. Public health law requires the hospital, clinical laboratory or physician to inform the local public health department of cases of notifiable diseases. In practice, this is an imperfect system with reporting completeness ranging from 30% for diseases such as whooping cough (pertussis) to 79% for HIV/AIDS[23]. There are no sanctions for failure to report. While there is a national list, each state has its own variant. In a large metropolitan area, reporting is done at the local level; in more rural areas, contact will be with the state department of public health. Reporting to the public health departments occurs via a diversity of media, including telephone, FAX, and computer-based electronic transfers. Naturally occurring disease outbreaks are generally slow to develop, and the system evolved to operate at a correspondingly long time scale.

The advent of the recognized risk of biowarfare attacks, with the fear of more rapid progression, has driven efforts to automate the process of surveillance. It has been estimated that currently nearly half of public health professionals do not have a computer on their desk. The National Electronic Disease Surveillance System (www.cdc.gov/nedss), including the Laboratory Response Network[67] (LRN), is an evolving system of standards and specifications intended to expedite secure information transfer. Currently, prototype systems are being evaluated in a limited number of states. The LRN is presently automated in most states.

International surveillance efforts are coordinated by the World Health Organization[97], of which the CDC is a key component. Nation-to-nation information is exchanged via a network (partially automated).

Information flow from CDC back to caregiver usually is hierarchical from the CDC to state and from state to local institution. In the case of an acutely evolving crisis such as the anthrax attacks of Fall, 2001, the communication may be in parallel to both state and local levels.

The CDC maintains active response teams ready to respond to acute situations. They also maintain biological "stockpiles," each of which contains therapeutic support for 100,000 people. These packs can be deployed in less than 12 hours to assist in the response to an emergency[42].

4 A Cyber CDC

Given the magnitude of Internet-scale threats due to worms, we argue that it is imperative for the Internet in general, and for nations concerned with cyberwarfare in particular, to attempt to counter the immense risk. We envision that a nation might address the threat with the cyber equivalent of the Centers for Disease Control (discussed in Section 3.2), which we will term here the Cyber Centers for Disease Control, or CCDC.⁷

While current defenses against computer worms are quite inadequate, there is a considerable amount of research and development which could lead to significantly strengthened defenses (as discussed in later sections). As the research progresses, the CCDC can begin deploying sensors and analysis tools in order to monitor for potential outbreaks. Although in Section 10 we discuss the need for automated detection and response systems be constructed of broadly distributed components, there still ultimately needs to be high-level centralized monitoring to provide a general "situational awareness," to facilitate strategic decision-making and response, and to coordinate the many human-mediated analysis and recovery tasks.

It is likely critical that the CCDC be a publicly funded institution, rather than relying on the commercial sector to fill this roll. First, it is not clear that any commercial entity will find the CCDC role as fitting with a viable business model. Second, many CCDC tasks, such as auditing third party applications and directing research, are outside the purview of commercial institutions. Such activities are not profitable, but still must be performed to secure the network infrastructure. Finally, many anti-worm defenses benefit from as broad as possible deployment—such participation needs to be encouraged even when there is no immediate commercial benefit from the deployment.

We see the CCDC as having six roles:

- Identifying outbreaks.
- Rapidly analyzing pathogens.
- Fighting infections.
- Anticipating new vectors.
- Proactively devising detectors for new vectors.
- Resisting future threats.

In the remainder of this section, we discuss each of these in turn. Our aim is not to comprehensively examine each

⁷This name has the benefit of being memorable due to its echo of the well-known CDC; but also the risk of being perceived as glib, since along with the similarities there are also of course major differences between the problems faced by the CDC and by the CCDC.

role, but to sketch them broadly in order to provide a basis for future discussion and development. In subsequent sections we discuss in greater detail a number of possible avenues of research to support some of these roles.

4.1 Identifying outbreaks

To date Internet-scale worms have been identified primarily via informal email discussion on a few key mailing lists. This process takes hours at a minimum, too slow for even the "slower" of the rapidly-propagating worms. The use of mailing lists for identification also raises the possibility of an attacker targeting the mailing lists for denialof-service in conjunction with their main attack, which could greatly delay identification and a coordinated response.

CDC Task: develop robust communication mechanisms for gathering and coordinating "field information." Such mechanisms would likely be *(i)* decentralized, and *(ii)* span multiple communication mechanisms (e.g., Internet, cellular, pager, private line).

For flash worms, arguably *no* human-driven communication will suffice for adequate identification of an outbreak. **CDC Task:** sponsor research in automated mechanisms for detecting worms based on their traffic patterns; foster the deployment of a widespread set of sensors. The set of sensors must be sufficiently diverse or secret such that an attacker cannot design their worm to avoid them. This requirement may then call for the development of sensors that operate within the Internet backbone, as opposed to at individual sites.

4.2 Rapidly analyzing pathogens

Once a worm pathogen is identified, the next step is to understand (i) how it spreads and (ii) what it does in addition to spreading.

The first of these is likely easier than the second, because the spreading functionality—or at least a subset of it—will have manifested itself during the identification process. While understanding the pathogen's additional functionality is in principle impossible—since it requires solving the Halting Problem—it is important to keep in mind that the Halting Problem applies to analyzing *arbitrary* programs: on the other hand, there are classes of programs that are fully analyzable, as revealed by extensive past research in proving programmatic correctness.

CDC Task: procure and develop state-of-the-art program analysis tools, to assist an on-call group of experts. These tools would need to go beyond simple disassembly, with facilities for recognizing variants from a library of different algorithms and components from a variety of development toolkits, and also components from *previous worms*, which would be archived in detail by a CDC staff librarian.

The tools would also need to support rapid, distributed program annotation and simulation. Furthermore, the team would need access to a laboratory stocked with virtual machines capable of running or emulating widelyused operating systems with support for detailed execution monitoring. (Less widely-used systems do not pose much of a threat in regards to Internet-scale worms.) In addition, code coverage analysis tools coupled with sample execution of the pathogen could help identify unexecuted portions of the code, which in turn might reflect the pathogen's additional functionality, and thus merit detailed analysis. (Or such unused regions could simply reflect "chaff" added by the worm author to slow down the analysis; an "arms race" seems inevitable here.)

4.3 Fighting infections

Naturally, we would want the CDC to help as much as possible in retarding the progress or subsequent application of the worm.

CDC Task: establish mechanisms with which to propagate signatures describing how worms and their traffic can be detected and terminated or isolated, and deploy an accompanying body of *agents* that can then apply the mechanisms.

It is difficult to see how such a set of agents can be effective without either extremely broad deployment, or pervasive backbone deployment. Both approaches carry with them major research challenges in terms of coordination, authentication, and resilience in the presence of targeted attack. The policy issues regarding the actual deployment of such agents are likewise daunting—who controls the agents, who is required to host them, who is liable for collateral damage the agents induce, who maintains the agents and ensures their security and integrity?

4.4 Anticipating new vectors

We would want the CDC to not only be reactive, but also proactive: to identify incipient threats.

CDC Task: track the use of different applications in the Internet, to detect when previously unknown ones begin to appear in widespread use. Unfortunately, Internet applications sometimes can "explode" onto the scene, very rapidly growing from no use to comprising major traffic contributors [68]. Accordingly, tracking their onset is not a simple matter, but will require diligent analysis of network traffic statistics from a variety of sources, as well as monitoring fora in which various new applications are discussed (since some of them may have traffic patterns that are difficult to discern using conventional traffic monitoring variables such as TCP/UDP port numbers).

CDC Task: analyze the threat potential of new applications. How widely spread might their use become? How homogeneous are the clients and servers? What are likely exploit strategies for subverting the implementations? What are the application's native communication patterns?

We give a cursory example of such an analysis in Section B.

4.5 Proactively devising detectors

Once a new potential disease vector has been identified, we would then want to deploy analyzers that understand how the protocol functions, to have some hope of detecting contagion worms as they propagate.

For example, to our knowledge there is no *KaZaA* module (one specific to how *KaZaA* functions) available for network intrusion detection systems in use today. Without such a module, it would be exceedingly difficult to detect when *KaZaA* is being exploited to propagate a contagion worm.

CDC Task: foster the development of application analysis modules suitable for integration with the intrusion detection systems in use by the CDC's outbreakidentification elements.

4.6 Resisting future threats

Devising the means to live with an Internet periodically ravaged by flash or contagion worms is at best an uneasy equilibrium. The longer-term requirement is to shift the makeup of Internet applications such that they become much less amenable to abuse. For example, this may entail broader notions of sandboxing, type safety, and inherent limitations on the rate of creating connections and the volume of traffic transmitted over them.

CDC Task: foster research into resilient application design paradigms that (somehow) remain viable for adaptation by the commercial software industry, perhaps assisted by legislation or government policy.

4.7 How open?

A final basic issue regarding the CDC is to what degree should it operate in an open fashion. For example, during an outbreak the CDC could maintain a web site for use by the research community. Such an approach would allow many different people to contribute to the analysis of the outbreak and of the pathogen, perhaps adding invaluable insight and empirical data. This sort of coordination happens informally today, in part; but it is also the case that currently a variety of anti-viral and security companies analyze outbreaks independently, essentially competing to come out with a complete analysis first. This makes for potentially very inefficient use of a scarce resource, namely the highly specialized skill of analyzing pathogens.

A key question then is the cost of operating in an open fashion. First, doing so brings with it its own set of security issues, regarding authenticating purported information uploaded into the analysis database, and preventing an attacker from crippling the analysis effort by launching a side-attack targeting the system. Second, the attacker could monitor the progress made in understanding the worm, and perhaps gain insight into how it has spread beyond what they could directly gather for themselves, allowing them to better hone their attack. Third, some sources of potentially highly valuable empirical data might be reluctant to make their data available if doing so is to release it to the public at large.

Given these concerns, it seems likely that the CDC would pursue a "partially open" approach, in which subsets of information are made publicly available, and publicly-attained information is integrated into the CDC's internal analysis, but the information flow is scrutinized in both directions. Unfortunately, such scrutiny would surely involve manual assessment, and could greatly slow the collection of vital information.

A related question is how international in scope such a facility should be. A national facility is likely to have a simpler mission and clearer management and accountability. However, there are real benefits to an international approach to this problem; one's allies are awake and working while one sleeps. A worm released in the middle of the night in the US would be far more likely to receive intense early research and attention in Europe or Asia than in the US itself. Thus, at a minimum, national level CDCs are likely to need to maintain strong linkages with one another.

5 Vulnerability Prevention Defenses

We now turn to a assessment of the different research areas relating to countering the large-scale worm threat. The first fundamental research area we look at is hardening systems to make them more difficult to exploit. Here, "systems" refers to software, configuration, and network policies.

In this section, we give an overview of the different subareas related to such hardening. In this an subsequent sections, for each area we assign a letter grade reflecting the area's "potential": **A**'s reflect areas for which we recommend funding priority, usually due to either high potential (even if perhaps also high risk), or low cost and medium potential; **B**'s for areas that have either medium potential and significant cost or high risk, or low potential but not high cost; and **C**'s for areas that either have low potential, extremely high risk, or (the most common) already have significant government or commercial funding support.⁸

5.1 Programming Languages and Compilers

One of the most important factors in implementing reasonably secure systems is the choice of programming language. Over two decades after their discovery, stack and heap buffer overflow attacks still account for a significant plurality of reported exploits.Even highly robust applications such as Apache have contained exploitable buffer overflow bugs[82].

Such attacks are effectively only possible in languages such as C, C++, or assembly, which don't perform array bounds checks, ensure type safety, or provide strong module isolation. Most other major languages include facilities for memory safety. However, the installed base for C/C++ software is enormous, and the languages remain popular for their high performance, so migrating such applications to other languages is an immense undertaking. On the other hand, the sophistication of automated analysis and transformation of code continues to grow, with some promising applications to increasing the robustness of programs, and the ever-increasing raw power of CPUs makes the need to squeeze the utmost performance out of a programming language less pressing with time.

Safe C Dialects: grade, C, keywords, Active Area . Safe C dialects provide familiar programming environments while enforcing type- and memory-safety. For example, CCured[64] uses compile-time global type inference, combined with run-time checks and garbage collection, to enforce type- and memory-safety. Unlike many other safe-C dialects, the language changes are very minor, and the run-time overhead ranges from 0-150%. Furthermore, since CCured uses source-to-source translation, it is broadly applicable. However, unlike traditional C compiler flow analysis, it requires a complete $O(N^2)$ global analysis. It also breaks library compatibility by changing the data format and calling conventions to include type information. Cyclone[49] is another "safe" dialect which requires more runtime overhead in return for faster compilation times.

Future work should include extending these techniques to C++; creating automatic library interfaces for existing systems; porting such systems to further environments; and efforts to increase its adoption.

Software Fault Isolation: grade, **C**, keywords, **Active Area**. Software Fault Isolation (SFI)[96] creates memory-safe sandboxes for executing arbitrary code by modifying the assembly/object code. SFI works by modifying potentially unsafe assembly operations, such as memory access and control flow, to ensure that they are correct with regard to some specified invariants. Thus, a module in an SFI based system can only execute its

⁸The grades were arrived at by extensive debate among the authors, generally converging on a consensus among the four of us. While we inevitably brought our individual interests and biases to the discussion, we also endeavored to remain cognizant of these and identify them as such during our discussions.

own code and specific functions, and can only modify its own memory. This enables Java-like[16] sandboxing techniques for arbitrary code, including systems such as Active-X[59], at a claimed cost of less than 10% in execution time. Depending on the granularity employed, the resulting code may be susceptible to some overflow attacks which force jumps to *existing* code, but it is immune to all attacks which must inject code or violate the sandbox. Since it operates on the assembly level, it is effectively language neutral.

The biggest impediment is simply lack of availability of SFI-based systems. Colusa Software was developing such a system[84], combined with a portable virtual machine, when it was purchased by Microsoft in 1996. The SFI portions are currently unavailable, even for use in code produced by Microsoft.

StackGuard: grade, C, keywords, Active Area . StackGuard is a simple compiler calling-convention modification which prevents some classes of overflow attacks in C code[19]. StackGuard modifies the calling convention by first creating a random number (a "canary") and storing it in a randomly pre-selected section of memory. Then, on function entry, the return address is XORed with the canary before being stored on the stack. During the return process, the canary is again XORed to restore the return value. Unless the attacker can discover the canary, he cannot force a specific return value on the stack. This does not offer protection against overflows targeting function pointers or longjmp records, but works well against conventional stack attacks with almost no overhead. The approach has been adapted to include protection for other important resources, with canaries placed in front of function pointers or similar structures. Although incomplete protection, the extremely low overhead suggests that widespread adoption would improve the current situation substantially.

Nonexecutable Stacks and Heaps with Randomized Layout: grade, B, keywords, Mostly Engineering. Most programs written in unsafe languages do not need to generate code at run-time and can thus be run with nonexecutable stacks and heaps. Although nonexecutable stacks and heaps are a major improvement, an attacker may still overcome these by injecting calls to allowed library functions to accomplish their goals. Such attacks could be thwarted by randomizing data, library, and object layout during program loading and dynamic linking[34]. This approach results in no run-time overhead, with only minor overhead during the dynamic linking and loading process. To attack such a system, the attacker must be able to extract a considerable amount of data from the running program, without causing a crash, in order to generate a jump to the desired function. This could be made even more difficult by inserting guard pages, which will always generate an exception when accessed, during the randomization and relinking process.

To our knowledge, there has been no attempt to build such a complete system. Doing so would require significant modifications to the OS and compiler linking strategy and other components. Yet due to the very low overhead required, especially for persistent servers, it bears further investigation. Similarly, given a nonexecutable stack and heap, a static sandbox could be created to further restrict the library calls to those present in the original program.

Monitoring for Policy- and Semantics-Enforcement: grade, **B**, keywords, **Opportunities for Worm Specific Monitoring**. One form of host-based intrusion detection is to monitor a program's system call accesses and other features to ensure that the execution conforms with a model of expected execution. Previous systems have only considered system call patterns, which are vulnerable to a "mimicry" attack [95]. Future systems need to consider system call arguments and the program's execution trace when making an analysis.

A particularly powerful form of program monitoring is based on using static analysis of a program's source code to construct a model of its correct execution that can then be enforced at run-time[93]. Static analysis is very promising because of the ease with which it can be applied, coupled with the power of the resulting models. In particular, the technique developed in [93] results in a provably zero false positive rate—it detects many forms of attacks that alter a program's execution, none of which can arise from correct execution of the program.

This work should be extended to increase performance and precision. In addition, the ability to analyze assembly or object code to create "black box" models would be highly valuable, allowing this technique to be extended to programs without source code. The provably zero false positive rate is perhaps the most valuable feature, as it enables effective and instantaneous response.

Automatic Vulnerability Analysis: grade, B, keywords, Highly Difficult, Active Area . Program analysis

techniques attempt to prove various properties about applications, which can include safety and security properties. Thus, improved program analysis tools could be applied to source code or assembly code to produce whiteor black-box security analyzers and verifiers. Some work has already been conducted, including attempts to discover buffer overflows in C[94]. A related approach verifies that a program conforms with specified constraints, such as "integers from untrusted sources must be sanitized before use" or "the kernel should not dereference user-supplied pointers without first ensuring their validity" (taken from[2]).

Further work could extend these techniques to detect such errors at the assembly level, and to infer other properties about the code, such as whether the code requires executable stacks or heaps, or makes specific patterns of system calls.

5.2 Privilege Issues

Fine-grained Access Controls: grade, **C**, keywords, **Active Area**. Fine-grained mandatory access controls[1] offer another useful tool for building secure systems. By being able to specify the allowed capabilities of programs and users, one can prevent a compromised program from being able to damage the rest of the system. As an example, if a web server is not allowed to initiate outgoing connections, a worm running in the web server's address space cannot spread. Unfortunately, such systems are currently difficult to set up and not widely available. Considerable effort should be spent on integrating these techniques into commodity operating systems, and considerable research still needs to be done on making them generally usable and easy to configure.

Code Signing: grade, **C**, keywords, **Active Area**. Code signing uses public-key authentication to verify that the module in question was produced by the stated party. Although highly useful for auto-updaters, it provides no protection with regards to attacks and flaws targeting the authenticated code.

Privilege Isolation: grade, **C**, keywords, **Some Active Research**, **Difficult Problem**. Privilege isolation is a technique long known (for example, the Mach kernel[77] and Plan 9[70] kernels), but seldom employed. It works by separating a program into different parts which are isolated from one another except by interprocess communi-

cation. In its simple form, a program might be divided into a small, separate part which requires superuser privileges, and a larger part for the remaining functions. Thus, a compromise of the main program won't result in an escalation of privileges. (E.g., the recent OpenSSH flaw[72] was not exploitable when privilege isolation is employed.)

Unfortunately this technique is seldom employed, mostly due to usability concerns, as it requires restructuring software. It also does not help with attacks that do not require privilege escalation, which some worms do not. Thus, work should focus on what is required to make this technique more widely applicable.

5.3 Protocol Design

Protocol design can dramatically affect the ease with which which a worm can propagate, but guidelines and principles for worm-resistant protocols have not been developed. These should be developed and applied to new and existing protocols. Existing protocols should be changed where possible, and new protocols should be examined with the guidelines in mind.

Redesigning existing protocols may not be practical due to the size of the installed base. However, it might be possible to mandate some protocol redesigns before potentially dangerous protocols are used on selected corporate or governmental networks.

Design Principles for Worm-resistant Protocols: grade, **A**, keywords, **Difficult**, **low cost**, **high reward**. It is an open problem as to whether protocol security properties can be efficiently described, but it is critical that we attempt to do so. Although some properties such as automated forwarding of connections can be easily identified as poor choices, other, more subtle properties are elusive. The effect of layering protocols is also difficult to anticipate, but should be considered.

Proving Protocol and Application Properties: grade, **A**, keywords, **Difficult, high reward**. Related to describing worm-resistant properties is verifying those properties. It would be useful to verify properties for new and emerging protocols, and to automatically generate code that implements a protocol and is guaranteed to be resistant to attackers who manipulate the communication in undefined ways. It would further be useful to build an interpreter that detects programs which violate the protocol. This appears to be a particularly difficult area of research, but the rewards are substantial. Network security can be significantly enhanced by both verifying a protocol and ensuring that implementations automatically conform and validate input.

Distributed Minable Topology: grade, **A**, keywords, **Hard but Critical**. Worms can identify potential new targets using existing protocols and services available across the Internet. Not all of these need to provide information that could be used by worms.

One particularly dangerous source of information is available from metaservers. Metaservers act as matchmakers, returning a list of servers running a particular service. Since this list is effectively a hit-list, a worm could take considerable advantage of a metaserver to accelerate its spread. Services such as Google[40] are also metaservers, directing users to specific services of interest. In some cases, if the metaserver is able to efficiently map a subset of the network in order to estimate performance between the querier and potential servers, it no longer needs to return an entire list of servers, only the single server chosen to support the application. The metaserver can also track which requests come from which machines, using this information to detect anomalous query patterns.

Network Layout: grade, **C**, keywords, **Costly**. Topological networks are potentially highly vulnerable to worms, as the information contained on an infected machine can be immediately used to find new targets (Section 2.1.2). Furthermore, most such networks are composed of homogeneous components. If, however, the network is made up of multiple component types, which can never co-occur, then a single-exploit worm can't traverse the network. Components can refer to hardware or software elements, from which split networks can be developed.

An example would have networks composed of Web servers and clients. If servers never directly talk to other servers, and clients never directly talk to other clients, then a topological worm (say one that inspects client histories and server logs to find other elements in the network) can't propagate through the network unless it has both a server exploit and a client exploit. In practice, however, the Web does not have this structure. Servers can talk to servers (because proxies and caches are both servers and clients), and potentially the same machine running a server can also run a client (for example, any desktop machine that runs a server will generally have this property).

The constraint of having two different types of components, neither of which directly talks to its own kind, can be expressed abstractly as requiring that the graph of the network be two-colorable, where the two colors will reflect the two types of components (client and server, in our example above).

Two major concerns with this approach are cost and flexibility. First, it requires two distinct software implementations. Thus, it will be very expensive to apply to mass-market programs like KaZaA. Second, the bipartite requirement can impede efficient operation of the application. For example, proxies and caches would require an indirection step in order to maintain the 2-color property.

5.4 Network Provider Practices

There are several important practices which are not in place today but could, if widely deployed, mitigate the damage an attacker can inflict upon the general Internet. It may be appropriate to consider mandates to deploy and further develop these techniques.

Machine Removal: grade, C, keywords, Already Under Development. Currently there is no standard protocol to notify ISPs that machines are compromised and that they should be removed from the Internet. This enables attackers to continue to launch attacks even after initial detection. Given the possibility of attackers using worms to compromise large numbers of machines, it is critical that these machines be removed in an efficient and secure manner.

However, providing such a standard mechanism must also include addressing the problem of ensuring that the mechanism not itself be used to impair victims. Related to this, significant liability issues must also be resolved before an ISP would be willing to deploy it. Finally, the efficacy of the approach depends on the speed and thoroughness with which compromised machines are detected. If only a small proportion can be removed, that will provide little practical benefit.

5.5 Implementation Diversity

While systems with a diverse set of software implementations don't prevent attackers from developing exploits, they do gain significant protection against large-scale worms and similar threats. The speed of active-scanning worms is a function of the size of the vulnerable population, so if fewer machines are vulnerable, not only can fewer machines be compromised, but it takes longer to compromise the vulnerable population. A good example of this phenomenon is the Scalper worm[60], which has spread slowly. While its slow spread may in part be due to its scanning routines, a great deal of the slowness is because it only targets FreeBSD systems running Apache, and this is not a large population.

Unfortunately, computing environments tend to form *monocultures* for reasons of compatibility, inertia, ease of administration, and deliberate corporate efforts to develop and maintain software monopolies (per Section 2.5). Monocultures represent a dangerous phenomena, as bugs in homogeneous systems are much easier for an attacker to exploit.

5.6 Synthetic Polycultures

Synthetic polycultures. grade, **C**, keywords, **Difficult**, **may add unpredictability**. One area of potential research would be techniques to develop synthetic polycultures: can binaries be transformed to create synthetic diversity, preventing exploits from working on a large subset of the population[34]?

Simple manipulation of library linking is insufficient to do so if the attacker can insert code into the running application, as the code could simply search the space for the proper location. One possibility might be developing techniques to would make the code injection more difficult by randomizing the stack positions, call frame size, and memory layout. This has the potential to increase the difficulty of constructing buffer overflow and similar attacks, but doesn't help once the attacker manages to inject the code or higher level attacks which rely on application functionality. Such techniques could examined using programs with known holes to see what gains can be achieved.

A more comprehensive solution is a code obfuscation: the program, with associated library interfaces, is placed through "one way" code transformations. An interesting question is how difficult the resulting code is to analyze, and whether performance would degrade. Another question is how to apply such obfuscations not just to individual programs but to the entire operating environment. Finally, maintaining such code might prove much more difficult than for non-obfuscated code, due to the varying effects of bugs and much more complex information required for high-level debuggers.

5.7 Economic and Social

Why Is Security Hard: grade, B, keywords, Active Area of Research. Even with the considerable tools still available, security and robustness are not common properties. There needs to be a solid understanding of why practices remain so poor. There are many economic and social reasons why good software practices are not commonly employed, ranging from cost to programmer inertia. In order to change these factors it is important to understand them in detail. Significant user and economic studies need to be conducted as a first step in a campaign to improve the general quality of applications and protocols.

6 Automatic Detection of Malicious Code

We can imagine extending current firewalls to accept a message from an Internet-level detector, warning that a worm is operating on the net and targeting a specific port. The firewall can then make local decisions to restrict access until human analysts can begin a recovery plan. Although any restriction can be short lived, it must be automatically imposed because of the expected speed of future worms. Similarly, host-based modules can act to restrict the actions on the host before damage is done. Any more sophisticated response system will also require detection techniques.

Thus a critical problem in creating automatic responses is accurately and automatically detecting and analyzing a worm's operation and correlating the information from numerous detectors to form automatic defenses. This section discusses new and existing detection strategies, while Section 10 discusses the problem of correlating the results in a distributed manner. Distributing the results can be accomplished with a broadcast or multicast network, though there are difficult issues concerning implosion, reliability, and congestion to work out; and some form of highlyscalable cryptographic techniques would be needed for authentication.

Detectors can either operate on the hosts, near the leaves of the network,⁹ or in the backbone, with decisions made either locally, at a centralized location, or in a hierarchical fashion. Whatever detectors employed need to be both highly sensitive and robust to inadvertent and deliberate false positives and negatives. Successful detectors will likely look for anomalies that different types of worms exhibit in their Internet traffic patterns (or on their victim machines) as a consequence of their replication techniques.

It is also highly beneficial for detectors to distribute information to all those who wish to receive notification, so subscription to the notification service will need to be available on attractive terms of some sort. One model would be for the sensors to operate as a governmentsponsored service intended to protect governmental computers from the direct and indirect effects of a worm, but the notifications available in an "Emergency Broadcasting System" style: as information freely available to whoever wants to use it, perhaps with some degree of dissemination mandated among Internet service providers.

6.1 Host-based Detectors

Host-based Worm Detection: grade, **A**, keywords,**Critical**. Some protocols such as peer-to-peer systems can support very fast topological worms which, at the micro level, appear as normal network traffic due to the query mechanisms and other features. Similarly, contagion worms can completely resist traffic-based analysis. As these applications continue to evolve to evade suppression, worms may become even harder to detect.¹⁰ Thus, to detect sophisticated contagion worms

we likely will increasingly have to rely on host-based intrusion detection techniques.

In such a scenario, the IDS needs to halt the program and distribute a warning to the peer-to-peer network. We could envision that when the number of warnings reach a critical mass, the peer-to-peer network would shut itself down, though clearly this would depend on the require a highly robust detector, and might be completely impractical depending on the trust model, criticality, and constituency of the users of the network.

Building a highly robust detector does not necessarily require building highly robust detectors for individual hosts. It might work for the host-based detector to forward a "measure of confidence" metric to an external system, which correlates the results from many such sources as a way of reducing false positives and increasing sensitivity. On the other hand, the worm might itself include mechanisms for flooding this external system with bogus reports in an attempt to confuse it, resulting in a "race" whose dynamics might be difficult to predict. Similarly, a highly robust detector may benefit from only being installed on a subset of hosts, although this may prove difficult due to sampling problems and coverage issues.

Existing Anti-virus Behavior Blocking: grade, **A**, keywords, **Critical**. Behavior blockingis an anti-virus technique which halts programs from performing certain actions deemed necessary for a virus to spread. Although potentially powerful, it has not been widely deployed due to usability concerns and false positives. The same techniques could be used to construct host-based sensors which rely on detecting necessary behaviors and report them to a correlator for further analysis. Again, such an approach might be able to leverage correlation across multiple sensors to diminish false positives.

Wormholes and Honeyfarms: grade, **A**, keywords, **Low Hanging Fruit**. A *honeypot*[73, 15, 76] is a machine whose only purpose is to be compromised by an attacker in order to detect and analyze the attacker's behavior. A distributed network of honeypots would form an excellent detector network except for the machine cost, administration cost, and distributed trust needed to create such a system. The *honeyfarm* approach eliminates these costs while improving the precision. It is built us-

⁹Leaf node detectors can either rely on symmetric routing, be fed traffic from all network links, or involve cooperating detectors which can examine all traffic to and from the machines on the network leaf

¹⁰Current peer-to-peer networks have been under attack by legal challenges because of the prevalence of copyright violations. Since the latest tactics involve legal threats against the users who distribute larger quantities of content and traffic shapers to reduce the available bandwidth,

future peer-to-peer networks will likely include significant anonymizing techniques which will make network detection of worms which attack these networks much more difficult.

ing *wormholes*: traffic redirectors that tunnel traffic from disparate locations around the network to the honeyfarm. The honeyfarm uses "virtual machine" technology[92] to create the illusion of thousands of vulnerable machines using only a few actual systems. Once a honeypot image is compromised, either by an attacker or an automated program, the communication is redirected to other honeypot images to quickly classify the threat. Wormholes can also be affixed to "network telescopes"[61], to monitor even larger address ranges.

In order for an attacker to trigger false alerts from this detector, the honeyfarm itself must be compromised. A worm which avoids this detector must either be able to remotely determine that the wormholes don't represent actual machines, or that the compromised machine represents a honeypot and not an actual target. One significant disadvantage, though, is application-specificity: the system can only detect worms which target the cultured honeypots.

6.2 Network-level Detectors

Edge Network Detection: grade, A, keywords, Critical, Powerful. An infected machine that generates a large number of scans is detectable on its associated network links, as is a large amount of aggregate scanning. Since most normal traffic receives positive responses, while scans are met primarily with negative responses or nonresponses, this anomalous pattern of behavior can be easily detected when there exists symmetric routing. The counterpart, detecting incoming scanning, is far less reliable due to IP spoofing. Other propagation strategies may also show clearly anomalous behavior.

A suitably large network of cooperating detectors can thus use the presence of compromised machines to broadcast a warning to the rest of the net. Although there are significant trust issues when building such a network, these form potentially effective detectors with a reasonably limited deployment.

Backbone Level Detection: grade, **B**, keywords,**Hard**, **Difficult to Deploy**. By definition, active-scanning worms must scan the network in order to discover new targets. Although scanning occurs all the time in the public Internet, the increase in scanning from an active worm is sizeable, and can possibly be detected in the backbones. Other anomalies may also be detectable. One difficulty is that backbone routing is highly asymmetric, so simple traffic monitoring is insufficient to detect a worm's scanning behavior. An open question is whether a group of communicating backbone sensors, by using coordination and statistical inference, can detect the presence of a worm early enough to provide for a response.

This approach may be difficult due to the volume of communication involved. Such a sensor would also be highly robust, due to the high volume traffic monitored. One possibility is for each sensor to keep a record of all responses which occur within a few second time window. (See [83] for a discussion of techniques for remembering fingerprints of individual packets seen by a high-speed router.) Some subset of the requests are broadcast to all other sensors, who respond as to whether the responses were noticed. This can be used to gain a statistical estimation of the current level of scanning occurring on the network.

6.3 Correlation of Results

Single point detection is often insufficient to determine that a worm is operating. Instead, the results of many sensors may be required before a particular entity is confident to act on the conclusion that a worm is operating. Similarly, different sensors are required to detect different worm types. Thus it is important to correlate and combine information from multiple sensors of various levels of trust.

Conventional Centralized Results Correlation: grade, **B**, keywords, Some commercial work. A central coordination service accepts information from sensors distributed around the net, summarizes the results, makes conclusions, and reports the results using automatic and manual systems. Such systems are already forming for general alerts, such as Security Focus's ARIS threat management system[31], but are not being designed with automated mitigation and response. Such correlation strategies naturally apply to the merging of information from Internet-wide sensors to detect a worm. The coordination service needs to trust the sensors as a whole, while assuming that some sensors may be corrupted. Those systems which initiate responses need to trust the coordination service.

The significant concerns are the single point of failure and attack induced by centralization and the economic costs of building a central correlation service. Since the best defense occurs when all relevant parties respond to worm threats, such services need to be available to all to maximize their ability to protect the net, not just those who are willing to pay subscription fees.

Distributed Correlation: grade, **A**, keywords,**Powerful**, **Flexible**. In a distributed correlation system, the sensors broadcast their results to all subscribers or to a local verifier which possibly aggregates and then relays the alerts. This provides each subscriber with access to the low-level data, enabling local policy decisions, which could factor in different trust levels for various sensors.

The distributed coordination approach adds communication overhead, which may be a significant concern, particularly if the attacker can create "chaff" that stresses the communication channels. Distributed coordination also requires more processing power, as some results may be redundant, and may require additional public key infrastructure, if the low-level alerts are transmitted across trust boundaries.

Indeed, the greatest concerns are the issues of trust: each individual entity in the decision network is potentially untrustworthy, so the protocol must resist entities that try to disrupt the network, create false positives, or create false negatives. This could become quite challenging if a worm manages to grow to a large size and attacks the coordination network by flooding it with bogus messages. More details of the requirements for such sensors are in Section 10.

Worm Traceback: grade, **A**, keywords,**High Risk**, **High Payoff**. Related to the problem of worm detection is that of worm *traceback*: given that we have detected a worm, can we ascertain from where it was originally launched? Worm traceback has received essentially no attention to date in the research community. With certain kinds of worms, such as random scanning worms, it may be fairly feasible to detect the first instance through the use of network telescopes[61], due to the fact that the first copy of the worm is likely to scan for quite some time before finding the second infectable host, and the scans will likely impinge on the address space of a sufficiently large network telescope before finding a victim. If the random number generator of a worm is reversible, this may also help (though a well-designed worm could easily avoid this mistake). For more sophisticated spread designs, it is quite unclear what the best approach would be. However, given a widespread and effective detection infrastructure, there are likely to be some clues left in the earliest signs of whatever the worm footprint turns out to be. This problem is likely to be challenging, but is obviously of the highest importance in deterring future worm attacks and, in some contexts, for gauging appropriate responses.

7 Automated Responses to Malicious Code

Since novel malware can spread much faster than humans can analyze or respond to it, a successful defense against such worms must be automated. The defense needn't necessarily be perfect; a successful automated response could slow down a worm enough to make human response relevant. In this section, we discuss possible research efforts along these lines.

Due to the speed of such malware, such response can be extremely challenging: a natural tendency is to err on the side of being "trigger happy," but the consequences of such response could themselves prove very damaging, and indeed an attacker might attempt to trigger the response mechanism as a way of impairing the victim rather than using a worm against the victim directly. Thus, the decision to respond must be highly robust against false positives and manipulation. One possibility is the use of cascading defenses, where prohibitions become more restrictive as more systems are compromised, reaching an equilibrium between the aggressiveness of the worm and the strength of the response; we discuss a "cell-based" model that permits graduated response below.

Host-Based Response: grade, B, keywords, Overlaps with Personal Firewall. It is an open question whether one could develop programs which could respond to "worm on the loose" alerts in an intelligent manner in order to protect the local host, beyond what could already be achieved using network-based responses. One advantage of host-based techniques is that the responses can be considerably more selective, especially if any warning includes information on susceptible versions. Thus a web server would only defensively respond if the warning suggested it was vulnerable. A disadvantage is that it would require considerably wider deployment to achieve a noticeable effect on the overall speed and spread of a worm.

Edge Network Responses: grade, A, keywords, Powerful, Flexible. It should be possible to construct filters which, when alerted, automatically filter classes of traffic. The problem is constructing responses which don't adversely affect normal traffic, or at least limit the damage during the filtering process. Once a worm is detected to be operating in the network, one can envision network devices which filter out traffic that reflects behavior corresponding to the worm's propagation mechanism. An example would be the response to a port 80 (HTTP) worm which would temporarily filter incoming web traffic until human analysis can determine sufficient defensive measures to render one's systems immune. Simply blocking particular IPs is not sufficient due to a worm's behavior, where new machines are infected at a rapid rate.

One facet of the research is attempting to devise prohibitions which either have no effect on normal traffic or a minimal impact on critical traffic. Clearly, different prohibitions will work best for different classes of worms.

More sophisticated response could begin by proactively mapping the local network using nmap-like[38] techniques to understand its components and topology. If an alert contains concrete information about potential vulnerabilities, the response could be tailored to only interrupt traffic which could be a vector for infection, based on the local site information.

Such responses could also be spread throughout an internal network, to create a containment system.

Backbone/ISP Level Response: grade, **B**, keywords, **Difficult, Deployment Issues**. Some responses can easily be envisioned by the ISP, such as limitations on outbound scanning by infected machines (which slows the spread of scanning worms). An important question is whether more sophisticated responses could prevent or mitigate other attacks and other strategies. ISP responses have a significant advantage in protecting more machines with a single response. Additionally, ISPs are at a good location to construct defenses which eliminate the outbound spread of an attack, as part of a general ISP program of responsibility. However, there is a very significant potential legal disadvantage, as now the ISPs may be responsible for worms which evade their defenses, as well as for any overreactions that impair legitimate traffic. The cost might also limit deployment.

The biggest concern is that ISP responses need to be at a "least common denominator" level, as responses which may inadvertently affect customers may never be employed, even if they could stop a worm's behavior.

National Boundaries: grade, **C**, keywords, **Too Coarse Grained**. Although it might seem attractive, a national boundary is likely not an effective location to conduct a meaningful response. The main problem is that it will be too easy for the attacker to either evade the boundary (for example, by first infecting an ally of the nation's, for which the boundary is not in effect), or simply to have already seeded the infection within the nation before the worm is detected.

However, it is possible that such national boundaries will be constructed to meet other forms of cyber threats, in which case adding a worm suppression mechanism may be relatively cheap and would add a layer (albeit brittle) of defense-in-depth. Also, while not preventing the spread of the worm, the boundary might serve to slow it somewhat (depending on its spreading mechanisms), and could also somewhat impede control of the worm after it has spread within the nation.

Graceful Degradation and Containment: grade, **B**, keywords, **Mostly Engineering**. An important property of many successful worm defenses would be that they fail gracefully: the ability to contain an infection and keep it from spreading after some set of machines have been compromised. Good defenses should have the ability to quarantine sections of users or networks to prevent the attack from spreading. This is critical if one wishes to contain worms operating in a corporate environment. A related question is then whether the defenses can recognize that a minimal mitigation strategy is ineffective and respond by imposing more severe prohibitions.

A promising possibility is a "many-unit containment" model. In this approach, a local network is divided up into many cells, with each cell able to determine that a worm may be operating within it, and with secure communication to all other cells. A single infected cell would simply attempt to quarantine the infected machine on the particular port. If two cells are infected, then perhaps all infected cells quarantine that port. If three cells are infected, all cells begin restricting incoming traffic except for a predetermined white-list. Other thresholds or strategies could of course be defined.

Such a system offers many benefits: it quarantines infections into small groups, hopefully containing the infection. More importantly, it offers graceful degradation of protection and network performance. As the infection continues to spread, the system can increase the severity of response. By eventually halting all traffic, it is able to completely contain an infection, but hopefully can respond before that. The major disadvantage is that it requires numerous devices across the corporate intranet, anywhere from 10 to 100 or more, to perform the cell by cell isolation, which creates significant engineering problems.

Data formats for Worm Description: grade, B, keywords, Important, but requires more experience before proceeding. There is a need for research on ways to describe worms, important parts of worms, and actions that have or might be taken in response to worms. For instance, correlator logs will need to describe the structure of the worm event. Data formats to describe how to recognize an infection vector connection in progress over the network would be valuable. Ways to describe signs of an infection on a host would be useful. Canonical descriptions for the set of network addresses potentially vulnerable to a worm could be valuable. Finally, response actions could be described, including sets of addresses to block, conditional conditions for when to block certain kinds of connection, and actions to take on hosts, etc. If an adequate understanding is gained of which of these things are feasible to describe well and generally, then it may be useful to promote standardization of them as a technical transfer solution for this research.

8 Aids to Manual Analysis of Malicious Code

Presently, most malicious code is analyzed by humans employing conventional debugging and disassembling tools. For example, Code Red I was disassembled and analyzed by Ryan Permeh and Mark Maiffret of Eeye digital security in about 24 hours, and Nimda was disassembled and analyzed by Ryan Russell of Security Focus in about 40 hours. Virus detection companies routinely require two weeks to develop detection algorithms for complicated viruses. For complicated, fast-moving worms, propagation to all available victims can be completed prior to human analysis.

Collaborative Code Analysis Tools: grade **A**, keywords, **Scaling is important, some ongoing research**. The current disassembly and analysis tools generally do not scale beyond a couple of individuals. Considerable research has been employed in improving collaborative programming process; a similar focus on the analysis process should improve the ability to analyze and understand how malicious code operates. Since understanding the operation of malicious code can be crucial in developing a response, it is critical that this step be shortened. It is an open question as to what new tools could enable greater cooperation in order to shorten this analysis problem.

Higher Level Analysis: grade, **B**, keywords, **Important. Halting problem imposes limitations**. Currently there are no tools which can describe potential behavior for an arbitrary code sample. Complete, general analysis of an arbitrary program is impossible (as it reduces to the halting problem), but certain behaviors such as file erasement or DDOS payloads should be recognizable because they need to perform system calls to implement these payloads. Similarly, deliberate obfuscation to prevent static analysis is important.

Having such tools could improve human analysis by indicating particular regions or potential behaviors of interest in a large program. With some worms (such as Nimda) being of substantial size, it is critical that human analysts are not bogged down by volume when searching for specific side effects.

Hybrid Static-Dynamic Analysis: grade, **A**, keywords, **Hard but Valuable**. While it is easy to build software that cannot be interpreted using static or dynamic technique in isolation, it is more difficult to develop software that is opaque to both static and dynamic techniques. Static and dynamic analysis can be used to simultaneously examine, interpret and trigger software, to aid in understanding code that is obfuscated or encrypted.

Visualization: grade, **B**, keywords, **Mostly Educational Value**. As a worm spreads, it may be useful to offer real-time analysis based on sensors to determine its behavior. Visualization tools, which could create results such as those seen in CAIDA's Code Red movie[6], may provide important insights. It is an open question as to what information can be gathered and how it could be presented in a useful manner. In many cases, visualization tools provide good sources for educational material without contributing significant research content.

9 Aids to Recovery

Another important concern is how to construct recovery systems which can operate after an attack, to speed the recovery rate and reduce the impact a worm would present on day to day operations.

Anti-worms: grade, **C**, keywords, **Impractical**, **Illegal**. Anti-worms, or "white" worms, are worms which act to remove security holes and other worms. They seem like attractive recovery mechanisms but there are significant limitations which make them impractical. The first is potential legal liability for any damage the anti-worm causes. Even a nondamaging worm, released to the wild, is a criminal act in many jurisdictions.

A second problem is timeliness: an anti-worm and any patch it installs must both be tested before release, while an attacker only needs an exploit and doesn't necessarily require complete testing.¹¹ An anti-worm that does not patch but instead restricts host behavior (for example, disabling a vulnerable service) does not have this problem, but in this case the potential for collateral damage or the attacker using the anti-worm response to further their own goals is immense.

A third limitation is the observation that many exploit toolkits remove the vulnerability used to compromise a system, and a sophisticated worm would undoubtedly perform a similar action. Thus, an anti-worm's author would need to discover a new vulnerability in order to correct compromised machines. However, we note that an anti-worm that spreads via a preconstructed hit-list might be able to beat the already-launched hostile worm, and thus avoid this difficulty only if the anti-worm author can quickly assemble a suitable anti-worm.

There have been at least three "white" worms: the Cheese[14] worm, which spreads by using the rootshell service created by some instances of the 1i0n worm[90]; Code Green[43], which scanned for Code Red II holes; and CRClean[50], which responded to Code Red II attacks. The latter two were released as source code, and it

is not clear if instances of the worms were released into the wild. All three anti-worms could only spread because the worms they targeted opened general security holes (backdoors) instead of closing them.

With worms such as Hybris[28] installing cryptographic backdoors instead of open backdoors, it is unlikely that a white worm could be written to displace a major attack unless it leveraged a separate vulnerability or used *flash* spreading via its own hit-list in an attempt to counter a particularly slow worm.

Patch Distribution in a Hostile Environment: grade, C, keywords, **Already Evolving Commercially**. An attacker who uses a worm to compromise a large number of hosts could program the worm to DOS major communication channels used by those who would respond to the worm. This could include automated channels used by preprogrammed responses, communication channels such as email lists, and patch distribution channels such as vendor web sites. An attacker could further be monitoring public channels as a guide to the best places to disrupt. Of critical importance is the development of a general patch and update distribution system which is highly resistant to wide-scale DOS attacks.

One possibility is a carefully constructed peer-to-peer system, where each vendor has a cryptographic key used to sign their patches. Patches which are properly signed and presented to a node in this system are spread to all nodes in the system, while each node responds to data requests only from local machines. A side effect is that such a network would automatically create a distributed system which could handle high loads, while reducing backbone bandwidth expenses. The addresses of these servers would need to be prefetched into DNS caches to allow them to be accessed if the root DNS has failed due to an attack. A Content Delivery Network, such as Akamai, already has many of these properties, and can potentially be modified to distribute patches in the event of a major attack.

Updating in a Hostile Environment: grade, **C**, keywords, **Hard engineering, already evolving**. Currently, an aggressive worm which infects the default install¹² could be particularly difficult for individuals to recover from as reinstalled machines could be reinfected before appropriate patches are applied. This is especially true

¹¹Code Red I was tested at least in part in the field, with one and perhaps two buggy initial releases which failed to spread effectively.

¹²The default initial configuration when a system is installed

if the worm make significant efforts to resist removal by prohibiting new code from operating on the machine and disabling anti-virus programs, thus requiring a reinstallation or booting from separate media to purge the infestation. For a large institution, such individual, per machine attention is a significant cost.

It may be possible to employ virus-derived techniques such as metamorphic code to insert a small bootstrap program[51] early in the operating system's install process. This program would resist removal by employing the same techniques which viruses use to avoid detection, while resisting behavior blocks by being installed at a very low level in the operating system. This program would be used to contact a central server to download any needed update and recovery code, without the need to use bootstrap media.

10 Coordination and Scaling

Many technologies for dealing with worms benefit greatly from cooperation and information sharing, due to several factors: the global slowdown achieved from a partial response; the fact that many sensors only produce results after infection; and that other sensors may have a high uncorrelated false positive rate which can be reduced through aggregation. Similarly, it may be much more difficult for an attacker to generate a malicious false positive in the face of coordinated data analysis.

Instead of describing a series of possible solutions, as common in the other sections, we outline the engineering and technical challenges needed to create a robust decision making system. There are two natural approaches: a central coordination system where all actors communicate through a single or small number of trusted analyzers, and a distributed system where every node has a more equal responsibility.

The centralized administration model, although attractive, needs to be trusted by everyone involved in the network. Additionally, any centralized system may be vulnerable to attack. The engineering challenges in creating a centralized system are also reasonably well understood.

A distributed system has compelling advantages, such as the lack of a single point of failure. Instead, large numbers of failures can be gracefully tolerated. Such failure modes potentially enable the system to be built with considerably lower-cost, lower-reliability components. This is especially true for devices which are not inline on the packet-forwarding path, since their failure will therefore not cause a disruption in network service.

A distributed system also benefits from heterogeneity. Supporting multiple sensors, analysis implementations, and responders within a common distributed framework increases the difficulty involved in creating worms which attack the response and analysis framework. Any attacker would need to corrupt multiple types of systems in order to render the framework inoperable.

Finally, a distributed system can potentially scale effectively from enterprise-level deployments to the entire Internet. If constructed correctly, the distributed system allows an internal deployment to both benefit, and share information with, the general Internet, and the Internet system can be derived from the same components and design.

One possible system consists of four primary agents: sensors, analyzers, responders, and aggregators. Sensors are concerned with detecting particular anomalies which suggest that a worm or compromise has occurred. Analyzers manage some of the communication between other components, aggregating messages for efficiency reasons, and also make local decisions as to the presence and virulence of a worm. Responders perform the interdiction needed to stop the worm, such as blocking ports, disrupting traffic, or changing network configurations. Aggregators act to cross trust domains, such as between institutions and the general Internet, and to reduce traffic by coalescing local analysis into higher-level abstractions. Naturally, multiple functionality could be combined into a single system.

One engineering and research obstacle is devising a protocol which these agents can use for communication. Such a protocol needs to be both simple to implement and easily extended. The first decision is communication: within a single domain of trust, any sensor messages are sent to trusted analyzers, while each responder must receive the results of analysis. Aggregators take all local data and perform some analysis, before presenting it to the general Internet, with corresponding data going the other way. Every message needs to be verified.

Although the communication itself is already challenging (as it involves multiple communication, reliable broadcasts, and considerable public key infrastructure), the more severe difficulties occur in developing the message structure, as discussed earlier in Section 7. The message from the sensors to the evaluators needs to both convey a general portion defining the application, degree of suspicion, and any potential information about the operating system. Additionally, a sensor-specific payload may be required to convey additional information. Similar information is needed for the messages between analyzers and responders, and for aggregating information.

The criteria is to create a protocol where the information is accessible and understandable to all programs participating, whether or not they understand the specifics of the source. Yet if the program understands the source's additions, this can provide valuable further information to foster better analysis.

Another engineering challenge lies in constructing analysis programs which can accept data from unknown sensors (i.e., relying simply on the sensor's judgment), known sensors (thus with more information about what the sensor actually detected, and with a higher degree of trust), and Internet-level aggregate information to determine whether a worm is active and how virulent its activity is. This analysis must also include an understanding of what responses have already been employed in order to gauge whether initial responses are adequate at stopping the worm.

A third challenge is how to deal with bad information. Although information within an institution may be assumed trustworthy (or at least reasonably trustworthy), information from the Internet is generally untrustworthy. One strategy involves isolating systems which are determined to have "cried wolf", and ignoring them in the future. Combined with only trusting Internet results when many individual players issue a warning (though bearing in mind that a worm can arrange to mimic this, once it grows large enough), this should allow results from the general Internet to be trusted, without trusting any individual or small group of systems.

A fourth challenge involves integration with existing systems. There are already numerous intrusion detection, anti-virus, email filters, and other defenses which all may record anomalies while a worm is spreading. It is important to understand how to integrate these devices as sensors, analyzers, and responders with minimal effort.

A fifth challenge is to make this system robust to attack. A small collection of agents should be removable without creating an adverse effect, and the agents should not need to depend on DNS or similar services in an emergency. Similarly, the system must benefit from heterogeneity in order to resist topological or other worms which might be designed to infect the response system.

A final challenge is the construction of visualization tools which can use this distributed information and present meaningful information for human analysis. Considerable research is needed both in how to present the information and to extract such information from the state of the distributed system, as discussed in Section 8.

11 Policy Considerations

Although this document is primarily intended to define the technical research agenda rather than considering policy issues, some policy issues unavoidably interact with the technical issues. In this section, we will outline those issues and the way they affect technical considerations.

Privacy and Data Analysis: Many sensors will require monitoring and potentially recording large amounts of information. Some of this information (including detected software configurations, traffic data, and potential payload) is potentially sensitive but may need to be monitored to create a relevant response. Of significant concern would be any sensors whose deployment is mandated.

In order for such sensors to be deployed, they need to be carefully constructed and disclosed so that individuals are confident about the limits of what data is collected and how it is managed. Issues such as exposure via subpoena or law enforcement search will require careful consideration during the development and construction of any such system; the degree of such potential exposure may significantly limit the ability to deploy sensors in some environments.

Obscurity: In many theoretical formulations of computer security, security should not *rely* on hiding the basic structure of the system. However there are benefits from obscuring some important information by increasing the risk and difficulty an attacker incurs in discovering the deployment of the sensors, thresholds of response, and other parameters.

During the construction and deployment of such systems, it is important to understand which portions need to be kept confidential and what can be safely disclosed. Ideally, items which must be kept confidential must not introduce privacy concerns among those who need to trust its deployment. This may not always be possible, however; when it isn't, a major policy struggle will likely emerge. Similarly, secrecy has limitations because it can be compromised by an insider, so it should not be a key feature of any defense, but a secondary factor to further confound an attacker.

Internet Sanitation: Some defenses, such as scan limiters, are best deployed by ISPs. However, these defenses do not directly benefit the ISPs or their customers, but only the rest of the Internet.

In order to deploy such defenses, there will need to be legal mandates or similar restrictions. Yet such deployments will need to be considered carefully: the devices need to be low cost, highly reliable, and generally trustworthy.

The "Closed" Alternative: An alternative way to prevent worms attacks are significant topological changes and restrictions: limit which machines may talk to others, define a fixed set of protocols which are allowed, and eliminate a considerable degree of flexibility.

As an example, suppose we could eliminate SSH clients from all machines but the users' desktops, and these would not run SSH servers. Such a restriction would prevent an SSH server worm from spreading; but it would also removes a large amount of functionality from SSH, namely the ability to forward authentication.

12 Validation and Challenge Problems

One difficulty in constructing defenses is that of testing and evaluating the results of systems: how can one ensure that the resulting system works, and is robust to attack? Beyond the conventional approaches of Red Teaming and the detailed testing of components, it may be necessary to directly model and evaluate the effects of an Internet-scale attack.

Similarly, a corresponding research program should have concrete milestones for detection, response, and analysis, as these represent the major problems in constructing a worm defense. Without the ability to detect a worm early in its spread, it is impossible to build meaningful defenses. Without the ability to construct meaningful automatic responses, a worm can perform its damage before human-based systems can respond. And without meaningful automatic and manual analysis, it may be impossible to determine what a worm author's objectives are and whether there are remaining surprises lurking in the worm's functionality.

12.1 Establishing a Common Evaluation Framework

The scope of possible worm attacks is very broad: different spreading mechanisms, topologies, target population demographics, ease of firewall penetration, polymorphism to thwart signature specifications, degree of deviation from "normal" traffic patterns, and combinations of multiple modes and/or hybrid mechanisms. Consequently, defining concrete milestones also requires defining the *evaluation context* against which the milestones are assessed.

Given the number of variables, it is not at all obvious how to define meaningful evaluation contexts for which the corresponding milestones will indeed serve to advance the core research, as opposed to optimize-for-thebenchmark efforts. Thus we argue that, in parallel with the general research efforts, work should also proceed on establishing a common evaluation framework (realistic scenarios, and the precise meaning of the evaluation metrics) as soon as possible; and this work should be recognized as valid research in its own right, rather than an adjunct considered only in the evaluation phase of individual research efforts. An example of a similar effort is the DARPA intrusion detection evaluations [55, 56, 57] that focused on system accuracy, or the more recent evaluations that focused on system capacity[41]. These and other efforts attempted to evaluate systems after the systems were developed; a significant improvement would require that all program participants agree to being evaluated prior to starting work, and each help to define appropriate metrics. We provide some initial suggestions here to initiate the process.

Research on developing a common evaluation framework should include identifying the right abstractions to be used to parameterize an evaluation context. For example, what are the most important elements when characterizing target population demographics? Are they captured by \mathcal{N} , the number of total hosts in the network, and N, the number of vulnerable hosts? Is an additional parameter needed to describe the clustering of the vulnerable hosts or their connectivity and trust patterns? To what degree can we abstract topology away from population demographics?

For analysis, the manner in which malicious software is written and distributed can greatly affect the complexity of the analysis task. There are techniques that are known to make both static and dynamic analysis difficult to perform. The basic principle is that sequences of actions that can occur in different orders are difficult to analyze, because of the explosion of possible system states. For static analysis, the use of function pointers makes it difficult to determine which call graph will really be created at run time, so any technique that relies on identifying specific function call sequences (or exploring all possible graphs) will be either evadable or computationally intractable. A related technique is encode the control flow within a routine using dynamically-computed boolean expressions or state variables, again in order to explode the number of possibilities that static analysis must consider. Similarly, dynamic analysis cannot explore all of the possible system actions that could lead to a malicious act, so any approach that relies on such exploration will be intractable. The analysis evaluation framework needs to find a way to focus on analyzing security-relevant events at the proper level of abstraction, rather than on the more general problem of understanding arbitrary code.

Another important aspect of the evaluation framework is including a notion of *resources available to the attacker*. It would be highly helpful to understand which worm scenarios require a resource-rich attacker (say, for extensive testing, or advanced knowledge of population demographics or topology, or numerous zero-day exploits) versus which reflect attacks available to attackers with only modest resources.

A final comment: since the problem area is so young, it will be difficult to develop such abstractions (and then choose a range from them for the simulation contexts) without also tracking the evolution of the parallel research on worm propagation and detection mechanisms. Thus, the research program needs to ensure frequent communication between these parties.

Similarly, a cryptographic-like ethos should be fos-

tered, where it is considered highly acceptable to break the systems of others, yet there is no dishonor in having one's systems broken by others in the community. This cooperatively-competitive nature of the cryptographic community produces strong results by insuring that the resulting systems are highly resistant to attack as systems are often throughly reviewed.

12.2 Milestones for Detection

In order to construct automated responses, it is first necessary to detect that a worm is operating in the network. Thus developing and validating detection mechanisms must be a priority for any program designed to address this problem. Since there are only a limited number of strategies which a worm can use to find new targets, it is probably best to focus on detecting anomalies produced by these strategies, in either a host-independent or hostdependent manner. We present several milestones which can be used to evaluate detectors optimized for various classes of worms.

The sensitivity to a worm's presence is a key metric for evaluating detectors, as this directly affects how early a sensor can be used to stop the spread of a worm. It is measured as the percentage of vulnerable machines which need to be infected before the detector can, with *high probability*, discover that a worm is operating on the network. This sensitivity can be tested by the validation tools discussed in Section 12.5 or verified using simulation and analysis.

Yet it is critical that such detectors not generate inadvertent false positives, despite their high sensitivity. Any sensor which generates significant false positives in the absence of deliberate manipulation will probably be unusable, unless enough sensors can be combined to eliminate these effects. False positives may also be engineered by an attacker. In some cases, it may be possible to eliminate these entirely, but it is critical that an attacker must possess significant resources (such as already compromised machines within an institution) to generate a false positive.

Additionally, since detectors must necessarily feed information into response mechanisms, it is critical that these detectors be hard to distort: it should be difficult for an attacker to create a false positive to trigger a false alarm or a false negative to avoid triggering an alarm dur-

Target Selection	Today	1 year	2 years	3 years	5 years
Strategy					
	Detect at	Detect at	Detect at		
	10%, easy to	1%, difficult	$\ll 1\%$, difficult	Widely	
Scanning	create false	to create false	to create false	deployable	
	positives and	positives	positives and		
	negatives		negatives		
			Early deployment		
External			Detect at	Detect at	
Target		Detect at	5%, difficult	1%, difficult	Widely
Lists	None	10%	to create false	to create false	deployable
(Metaserver)			positives	positives and	
				negatives	
				Early deployment	
			Detect at	Detect at	
Local		Detect at	<10%, difficult	1%, difficult	Widely
Target	None	25%	to create false	to create false	deployable
Lists			positives	positives and	
(Topological)				negatives	
				Early deployment	
			Detect at	Detect at	
Passive	None	Detect at	10%, difficult	\ll 10%, difficult	Widely
(contagion)		any level	to create false	to create false	deployable
			negatives	positives or	
				negatives.	
				Early deployment	

Table 1: Possible milestones for detecting classes of worms

ing a real attack. This can be evaluated by Red Teaming or similar techniques.

Scanning: Scanning worms, such as Code Red[25], Nimda[12], and Scalper[60], must by definition scan for new targets by generating addresses and determining if they are vulnerable. Since most individual scans fail, this creates a significant anomaly which detectors can notice. Of the strategies for autonomously propagating worms, scanning appears the easiest to detect.

Currently, scanning worms can be detected by network telescopes[61], i.e., large unused address ranges. (Alternative approaches include email-lures and SMB-lures [35].) Since these ranges are unused, any traffic is, by definition, anomalous. Thus any global increase in scanning can be easily noticed, at a reasonably high sensitivity. Yet these sensors can easily be avoided by worms if the attacker knows their location, and can also be triggered by an attacker capable of creating spoofed packets.

At the end of the first year, prototype detectors for scanning worms should be significantly improved. It should be possible to construct sensors which will accurately detect a scanning worm when roughly 1% of the vulnerable population has been infected. Such defenses should also be difficult for an attacker to trigger a false positive.

At the end of the second year, refined detectors should be made more robust, able to resist more false positives and negatives, while endeavoring to increase sensitivity to detect infections when $\ll 1\%$ of the vulnerable population is infected. This should also involve reducing the cost of the detection systems, enabling low cost deployment, with initial deployment of some systems available for early adopters who wish to protect critical networks (or simply to contribute to the research effort).

The end of the third year should see cost-effective deployment of actual systems based on the developed technologies, by those who wish to protect themselves from scanning worms.

Externally Queried Target Lists: Metaserver worms, worms which primarily acquire their target by querying a separate server, have the potential to be very fast. They potentially present anomalies both in network traffic and in the query stream. There are currently no mechanisms to detect such worms.

At the end of the first year, prototype detectors which can crudely detect such worms should be testable, although possibly avoidable by an adversary. Again, these can be verified by constructing simulated worms in the testing frameworks outlined elsewhere.

At the end of the second year, sensitivity should be significantly improved, with the resulting detectors correspondingly harder to fool by intelligent attackers. The third year should represent continued refinement, with early deployment for critical systems. By the fifth year, systems should be commonly available for arbitrary metaservers to neutralize this threat. A significant advantage is that some sensors would operate on the metaservers, allowing single point deployments for some host populations.

Local Target Lists: Topological worms, which acquire their target information using information contained on the local machine, can spread extremely quickly. The anomalies presented by worms using this strategy may be quite subtle, however.

At the end of the first year, prototypes detectors which can crudely detect such worms should be testable. Ideally, these detectors will also detect worms which use static target lists (Flash worms) and externally queried target lists (Metaserver worms). Due to the difficulty of detecting these worms, even noticing when 25% of the target population is infected represents a reasonable first year goal.

Given the detectors developed in the first year, second year milestones should focus on creating improved results: increasing the sensitivity, making the sensors difficult to attack, and other improvements. The third year should see even more refinements and initial deployment, with systems commonly deployable by the 5th year.

Passive Worms: Contagion worms do not generate new traffic; instead they rely on user behavior to determine new targets. These worms are highly stealthy from a network point of view. Fortunately, they will often be slower, in which case detection can drive human analysis with time to respond.

At the end of the first year, the goal should be simply to determine that detection is possible, regardless of constraints. Due to the potential subtlety of such worms, it may require numerous correlations of very low level anomalies to determine that such a worm is active.

At the end of the second year, the techniques should be refined to where they can detect worms when <10% of the target population is corrupted. For passive worms that are slow enough that human timescales enable responses, it is better to minimize the ability for an attacker to create false negatives, as long as the generation of false positives is detectable and traceable.

The end of the third year should see these techniques refined to improve sensitivity and robustness to manipulation, with early deployment possible.

12.3 Milestones for Analysis

Goals: Analysis to protect against rapidly spreading worms will need to operate in at least two modes. In the first mode, a worm will have been captured and some behavior will have been observed. The worm and the behavior will be available to the analysis process. In this mode, the worm is assumed to be actively spreading, so we require the results of the analysis as rapidly as possible (seconds to minutes), and the analysis needs to suggest mitigation strategies. To determine these latter, the analysis process needs to examine the code, isolate the securityrelated portions, and then identify prerequisites for execution (to determine if a prerequisite could be quickly and safely removed from vulnerable systems), and both the spreading modalities and the propagation characteristics (to determine if firewalls or routers could be configured to prevent additional propagation).

In a second mode, a deep understanding of the malicious code is desired, with the intended result being a long-term approach to predicting and preventing unseen, similar worms, and/or to assess the full implications of a worm attack that we were unable to prevent. To attain this understanding, the analysis process would more deeply address the above listed items, and also identify: trigger conditions for latent malicious behavior; new, previouslyunseen techniques; and attribution or forensic characteristics. Furthermore, if the code is seen to target a specific host, application, or operating system, the analysis might suggest alternative ways that the attacker may be attempting to target that element, so that humans could investigate that possibility.

State of the Practice: Analysis is currently done by experienced personnel. There are several existing models that we can call upon. In the first model, virus detection company employees are interested in analyzing viruses to determine a signature or write specialized code that can be used to identify the virus (usually from a file on disk). The time to do this varies with the complexity of the virus. Signatures for simple, unobfuscated viruses can be deter-

mined in a few hours of time[63], with subsequent testing to compare a proposed signature against many gigabytes of the most common existing non-virus software requiring six or seven hours. More complicated viruses that are obfuscated or encrypted can require several weeks to analyze and determine a signature, and complete testing of the resulting signature against all known software and patch levels can require a full day, due to the increased complexity of the signature. Software to recover from these viruses requires additional time to develop and test.

In the second model, software is reverse engineered to determine how it works. Some software is extremely complex (e.g., Microsoft's software to implement the SMB protocol) and requires large numbers of resources to reverse engineer. Luckily, malicious code is a more restricted domain, and we have at least one example that measured how long it takes to analyze malicious code: in [71], analysts were tasked with examining an executable and fully explaining the purpose of the software and the implication of having it installed on a system-a blend of the rapid analysis and the slower, more complete analysis. For a relatively small binary (total size 200KB) that employed limited obfuscation techniques, it required on average 70 hours for the evaluated entrants to examine, understand, describe and propose mitigation strategies for the code. This time did not include the time required to detect, locate and identify the binary as malicious. For mitigating the effects of such an executable, 70 hours is acceptable for a manually activated attack-but is much too slow for an attack that spreads automatically.

Metrics: There are many metrics appropriate for evaluating the performance of tools to analyze malicious code. Among them are: (1) the accuracy of the analysis in terms of the probability of detecting a prerequisite, trigger, and propagation methodology vs. the probability of a false alarm; (2) the completeness of the analysis, in terms of the percentage of detected prerequisites, triggers, and propagation methodologies; (3) the speed of the analysis in achieving its results, including an indication of the degree to which the process can be parallelized or distributed; (4) the usability of a tool, if human interaction can help the system focus its results; and (5) the impact of analysis and a selected mitigation strategy on the computing environment (in cases where the analysis will be performed on a live system). Covering all of this space will require more time than a five-year program will allow; so some sampling of the space will be required.

Milestones: A research program could measure progress against a set of progressively more complex worm types. An initial program might begin with a test set consisting of several worms of varying types captured "in the wild." Tools would be built to analyze these and develop mitigation strategies for each. The worms would require different prerequisites, use different triggers, and propagate in different ways. They would serve as a baseline, selected at the start of the program and used throughout its lifetime. To ensure that researchers develop solutions more general than just targeting the benchmark worms, new worms (with new prerequisites, triggers, and spreading modalities) could be added each year. each would be measured in future years. These new worms would incorporate attributes from more sophisticated worms than have since been seen in the wild.

12.4 Detecting Targeted Worms

Although most attackers will probably use general worms which indiscriminately target machines, an attacker could instead construct a worm which only propagates for a few generations within a limited address space. Such a *targeted* worm could be used to limit collateral damage and information leakage in a targeted strike. A scenario for a targeted worm would include scenarios such as corporate espionage, where the attacker wishes to only affect a small area in order to avoid detection.

Since targeted worms are attacking smaller address spaces, their spread may take just a matter of seconds before the entire range has been examined. Thus sensors intending to stop them may need to be considerably faster responding. Worse, they present fewer anomalies when compared with general worms; also, since the population is much smaller, the worms may be able to afford to instead spread more *slowly*, to gain stealth.

Previous worms such as Nimda have preferentially targeted the intranet[12], so some types of defenses developed to stop Internet-wide worms should also apply to targeted worms. In particular, defenses which rely on separating an intranet into many isolatable pieces should apply equally well to halting targeted worms if these defenses don't require external information in order to make their response decisions. During the evaluation process, it is important to understand which defenses may also stop such targeted worms, and which are only suitable to halting Internetwide worms.

12.5 Tools for Validating Defenses

Worm Simulation Environments: grade, **A**, keywords, **Essential** Currently, there have been only ad-hoc simulators developed to evaluate the behavior of worms and defenses. In order to comprehensively evaluate defensive strategies and techniques, it is critical that larger, more realistic simulations be developed.

If one can construct a meaningfully detailed simulation of a worm on the Internet, then such a system could be used to estimate the secondary effects of worms and provide an environment in which to test defense strategies in face of attack. If suitably sophisticated, defenses could be either simulated or actually integrated into the simulator as separate devices, enabling comprehensive evaluation of their effectiveness. We note, however, that general Internet simulation remains an extremely challenging problem[30].

Internet Wide Worm Testbed: grade, **A**, keywords, **Essential** There have been several projects[52, 20] which rely on volunteers running programs on their own computers. This same technique could be used to test the behavior of a spreading worm on the Internet, either within a contained address range or across the entire net. It may be possible to recruit volunteers to deploy a simple daemon across the Internet. This daemon could simulate the behavior of a spreading worm.

Most strategies which an autonomous worm could use to spread across the Internet could be evaluated on the Internet with a simple daemon programmed to respond to a message and then replicate the spreading strategy, attempting to contact other daemons. If enough copies of the program are distributed, this can completely simulate the actual behavior of scanning worms. In order to increase leverage, some large "dark" address ranges could also be employed by using them to implement many worm models. Not only could this demonstrate a worm's behavior under real situations, but it may reveal particular infrastructure weaknesses which may be inadvertently affected by a widespread scanning worm. One serious difficulty with this approach is the monitoring alarms its traffic patterns might trigger at the various hosting sites, and the possible side-effects which the simulated traffic may create.

Testing in the Wild: grade, **A**, keywords, **Essential** There have been many minor worms seen in the wild, such as Slapper and Scalper, which can only infect a small number of machines. Other worms, such as Code Red and Nimda, still remain endemic on the Internet. The ability for worm detectors and responders to stop such worms can be tested by deploying the devices on the Internet, protecting otherwise vulnerable machines. The most significant limitation is due to the relatively unsophisticated nature of current wild worms.

13 Conclusion

Computer worms represent a significant threat to the United States computing infrastructure. A widespread attack could cause massive economic damage or be used in concert with a real world attack. Due to the significance of this treat, it is critical that defenses be developed in advance of a major attack.

There appear to be a limited number of strategies a worm could employ, which suggests that defenses which target strategies could provide meaningful protection. These strategies represent worms across the spectrum, from highly stealthy to extremely fast. Yet there are numerous attackers who could potentially employ such a worm.

Although our current defensive infrastructure is clearly inadequate, there are many areas in prevention, detection, response, analysis, and recovery which, if properly developed, offer substantial protection. We summarize the possibilities in Tables 2-6. Of particular interest are worm detectors, responders, and analysis tools.

We also advocate the development of a Cyber Centers for Disease Control, or CCDC, to coordinate both the research areas and the response to worms. A CCDC also has a significant prevention role, in analyzing applications.

Without sufficient investment, these technologies will not be developed before a major attack. With potential damages reaching over \$100 billion, it is critical that protective measures be developed in advance.

A Hardware devices

Many worm defenses will require widespread deployment within an institution, either at the computers themselves or embedded in the network, to form an effective defense. Although software has excellent economies-of-scale, the need to deploy defenses on most or all machines may prove prohibitive in some institutions, due to end-system heterogeneity and administrative burden. Thus it is attractive to consider network-level devices to detect and fight worms.

If the devices also generate anti-worm responses, many devices may be required in an institution. Each device can only prevent a worm from crossing through it, hopefully containing an infection on one side or the other. Thus, in order to provide a fine grain of containment, many devices are needed, unless the institution can keep the worm completely out.

Additionally, such devices benefit greatly from communication: a single device can detect a worm, triggering a general response. This response is best scaled by monitoring the magnitude of the threat, as more distinct machines are compromised and as more detectors report the presence of a worm, the response is magnified. This prevents both initial overreaction from crippling the network and an underreaction from allowing a worm to spread unimpeded.

A side effect of this strategy is that some groups of machines will be compromised. Increasing the number of devices deployed reduces the size of each group which may be compromised in an attack, providing the anti-worm response can cascade faster than the worm spreads. Thus it is critical that these devices be economically deployable throughout the site.

There are other requirements for such devices: they must effectively monitor and respond on gigabit networks, they need to be generally programmable, and they must be easy to install and administer.

Here is an example of one possible building block which meets these criteria. In 2004, this hardware platform should cost < \$500 to build, allowing deployment of devices costing \$1,000-5,000. There are other solutions using network processors which could also be employed, but the devise we discuss below is based on our experience with FPGA architectures. With these hardware systems costing considerably less than high-end PCs, this de-

Research Area	Grade	Keywords	Section
Protocols for Worm	A	Difficult, Low Cost,	5.3
Resistance		High Reward	
Proving Protocol Properties	A	Difficult, High Reward	5.3
Distributed Mine-able Topologies	A	Hard but Critical	5.3
Nonexecutable Stacks & Randomization	В	Mostly Engineering	5.1
Monitoring for Policy and	В	Opportunities for Worm	5.1
Semantics-Enforcement		Specific Monitoring	
Automatic Vulnerability Analysis	В	Highly Difficult, Active Area	5.1
Why Is Security Hard	В	Active Area	5.7
Safe C Dialects	C	Active Area	5.1
Software Fault Isolation	C	Active Area	5.1
StackGuard	C	Active Area	5.1
Fine Grained Access Control	C	Active Area	5.2
Code Signing	C	Active Area	5.2
Privilege Isolation	C	Active Area	5.2
Network Layout	C	Costly	5.3
Machine Removal	C	Already Under Development	5.4
Synthetic Polycultures	C	Difficult, may add unpredictability	5.6

Table 2: A summary of the research areas in prevention, ordered by grade and section

Research Area	Grade	Keywords	Section
Host Based Detectors	A	Critical	6.1
Existing Behavior Blocking	А	Critical	6.1
Wormholes and a Honeyfarm	А	Low Hanging Fruit	6.1
Edge Network Detectors	А	Critical, Powerful	6.2
Distributed Results Correlation	А	Powerful, Flexible	6.3
Worm Traceback	A	High Risk, High Payoff	6.3
Backbone Detectors	В	Hard, Difficult to Deploy	6.2
Central Results Correlation	В	Some Commercial Work	6.3

Table 3: A summary of the research areas in automatic detection, ordered by grade and section

Research Area	Grade	Keywords	Section
Edge Network Response	А	Powerful, Flexible	7
Host Based Response	В	Overlap with Personal Firewall	7
Backbone Level Response	В	Difficult, Deployment Issues	7
Graceful Degradation and Containment	В	Mostly Engineering	7
Data Formats for Worm Description	В	Important, More Experience	7
		Needed before proceeding	
National Level Response	C	Too Coarse Grained	7

Table 4: A summary of the research areas in automatic response, ordered by grade and section

Research Area	Grade	Keywords	Section
Collaborative Analysis Tools	A	Scaling Critical, Some Ongoing Research	8
Hybrid Static/Dynamic Analysis	A	Hard but Valuable	8
Higher Level Analysis	В	Important. Halting Problem Limited	8
Visualization	В	Mostly Educational Value	8
Anti-worms	C	Impractical, Illegal	9
Patch Distribution in a	C	Already Evolving Commercially	9
Hostile Environment			
Upgrading in a	C	Hard engineering	9
Hostile Environment		Already Evolving Commercially	

Table 5: A summary of the research areas in manual analysis and recovery, ordered by grade and section

Research Area	Grade	Keywords	Section
Worm Simulation Environments	A	Essential	12.5
Internet Wide Worm Testbed	A	Essential	12.5
Testing in the Wild	A	Essential	12.5

Table 6: A summary of the research areas in validation, ordered by grade and section

sign enables lower cost and wide deployment.

The proposed device centers around the latest FPGA families. The Xilinx Virtex 2 Pro series of FPGAs[47] combines an FPGA integrated with high speed network interfaces and 300 MHz PowerPC CPUs. These network interfaces support multiple protocols, including Gigabit Ethernet, Infiniband, and Fiber Channel.

One representative device is the Xilinx XC2VP7, which contains 1 CPU, 8 Gb/s links, 10,000 logic cells (4-LUTs with associated flip flops), and 792 Kb of memory. It is currently available in engineering samples, with pricing of approximately \$360 in Q1 2003 and \$180 in 2004 for large quantities.

The basic platform consists of one of these FPGAs, a slot for a single DDR SO-DIMM, 2 or 4 Gigabit Ethernet interfaces, and a small configuration store. Such devices should be constructible under the price target of \$1000 by the end of 2004. They are also highly programmable, containing both a significant microprocessor and a large amount of programmable logic. If the designer can ensure that most packets are processed in logic without processor interference, this device can easily maintain gigabit rates.

Just as conventional IDSs benefit from moving significant amounts of logic into the OS kernel to reduce kernel crossings and data being transferred, this approach requires placing the common logic in the hardware. As an example, if all but the SYNs and SYN/ACKs for TCP traffic, and all but the headers for UDP traffic are handled without touching the processor, this enables large data rates without tying up the processor. If the filtering suffices to avoid sending traffic up to the processor, this improves the data rate further, making Gb rates readily possible in such a low cost device.

This platform is also highly programmable. If 8 ports are available, this device could act as a low cost Gb router, a Gb IDS, or a Gb anomaly detector. More importantly, the programmable nature allows new algorithms and detectors to be implemented using the existing box.

B Example Worm Potential Evaluation: Half Life

One important feature of a Cyber CDC is to examine applications for their susceptibility to worm attacks. Although any network-aware software may contain flaws, some make more attractive hosts than others.

As an example evaluation, we consider the multiplayer game HalfLife[85]. Although released several years ago, HalfLife still remains popular on the Internet today, with over 20,000 servers and 85,000 players at a time[39]. Other video games in the first person shooter genre use a similar structure, so this analysis would apply to those applications as well.

This Windows game is split into two sections, a client which runs on the player's machine, providing the user interface, and a server which coordinates information and operates the game. To allow players to find a server, this game uses a metaserver to act as a matchmaker. Whenever a server starts, it connects to the metaserver and communicates appropriate information. When players wish to find a type of game, they contact the metaserver to find an appropriate server. In order that anyone can start a server, and to simplify coding, the server is included with the client software, with single player games using a server running on the local machine.

There are also stand-alone server programs, running on both Linux and Microsoft Windows.

Since there are usually only 20,000 servers operating at any particular time, a scanning worm would be generally ineffective, requiring many days to spread across the Internet. Similarly, each local server doesn't contain sufficient topological information to build a topological worm.

This application is susceptible to metaserver and contagion strategies, which could exploit various flaws in the client or server. A server side exploit could query the Gamespy[39] metaserver to obtain a list of servers running with various parameters. It then infects those servers, using the metaserver to find new targets. Such a worm would require roughly a minute to infect all servers.

A client side exploit could infect the client and then modify the local copy of the server. If the client ever initiates a public server, this modified server could infect other clients. Optionally, the server could be started and broadcast false information to attract new players: sacrificing some level of stealth in return for a significantly faster spread.

HalfLife is written largely in C/C++, and there have been buffer overflow vulnerabilities[5] reported in earlier versions of the software. Most players and servers use home machines equipped with broadband connections. Thus the machines themselves don't represent valuable targets, but they could be successfully used to conduct widespread DOS attacks.

Fast moving worms, either exploiting client or server side flaws, will produce a low grade anomaly to the metaserver for each compromised machine. For the metaserver spread, the anomaly is servers starting to perform queries to find other servers. For an accelerated contagion worm, the anomaly is an increased frequency in clients which start their own server. If the metaserver is actively looking for this behavior, it could correctly deduce that a worm is operating if enough anomalous activity is seen and temporarily halt the metaserver to stop the spread of the worm.

References

- [1] The National Security Agency. Security-Enhanced Linux, http://www.nsa.gov/selinux/.
- [2] Ken Ashcraft and Dawson Engler. Using Programmer-Written Compiler Extensions to Catch Security Holes. In *IEEE Symposium on Security and Privacy*, pages 143–159, May 2002.
- [3] Network Associates. Network Associates Corporate History, http://www.irconnect.com/net/pages/ history.html.
- [4] Avien. European Institute for Computer Anti-Virus Research, http://www.avien.org/.
- [5] Stan Bubrowski. Half life server overflows and format vulnerabilities, http://archives.neohapsis.com/ archives/bugtraq/2001-03/0111.html.
- [6] CAIDA. CAIDA Analysis of Code-Red, http:// www.caida.org/analysis/security/code-red/.
- [7] Cardcops. http://www.cardcops.com.
- [8] CDC. History of Quarantine, http://www.cdc.gov/ ncidod/dq/history.htm, May 2000.
- CERT. CERT Advisory CA-1999-04 Melissa Macro Virus, http://www.cert.org/advisories/ ca-1999-04.html.

- [10] CERT. CERT Advisory CA-2000-04 Love Letter Worm, http://www.cert.org/advisories/ca-2000-04.html.
- [11] CERT. CERT Advisory CA-2001-22 w32/Sircam Malicious Code, http://www.cert.org/advisories/ca-2001-22.html.
- [12] CERT. CERT Advisory CA-2001-26 Nimda Worm, http://www.cert.org/advisories/ca-2001-26.html.
- [13] CERT. CERT Advisory CA-2002-25 Integer Overflow in XDR Library, http://www.cert.org/ advisories/ca-2002-25.html.
- [14] CERT. CERT Incident Note IN-2001-05, http:// www.cert.org/incident_notes/IN-2001-05.html.
- [15] Fred Cohen and Associates. Deception ToolKit, http://www.all.net/dtk/.
- [16] Sun Computers. The Java Programming Language, http://java.sun.com.
- [17] ComputerWorld. Al-qaeda poses threat to net, http://www.computerworld.com/securitytopics/ security/story/0,10801,76150,00.html.
- [18] Counterpane. http://www.counterpane.com.
- [19] Crispan Cowan, Calton Pu, Dave Maier, Jonathan Walpole, Peat Bakke, Steve Beattie, Aaron Grier, Perry Wagle, Qian Zhang, and Heather Hinton. StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks. In *Proc. 7th USENIX Security Conference*, pages 63–78, San Antonio, Texas, jan 1998.
- [20] distributed.net. http://www.distributed.net.
- [21] David Dittrich. The Stacheldraht Distributed Denial of Service Attack Tool, http://staff.washington.edu/ dittrich/misc/stacheldraht.analysis.
- [22] David Dittrich. The Tribe Flood Network Distributed Denial of Service Attack Tool, http:// staff.washington.edu/dittrich/misc/tfn.analysis.

- [23] TJ Doyle, MK Glynn, and SL Groseclose. Completeness of notifiable infectious disease reporting in the United States: An analytical literature review. *AM J Epidemiology*, 155(9):866–874, May 2002.
- [24] Computer Economics. http://www. computereconomics.com/.
- [25] eEye Digital Security. .ida "Code Red" Worm, http://www.eeye.com/html/research/advisories/ al20010717.html.
- [26] EICAR. European Institute for Computer Anti-Virus Research, http://www.eicar.org.
- [27] Mark Eichin and Jon Rochlis. With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988. In *IEEE Computer Society Sympo*sium on Security and Privacy, 1989.
- [28] F-Secure. F-Secure Computer Virus Information Pages: Hybris, http://www.f-secure.com/v-descs/ hybris.shtml.
- [29] Peter Ferrie. W32//Klez, http://toronto.virusbtn. com/magazine/archives/200207/klez.xml.
- [30] Sally Floyd and Vern Paxson. Difficulties in Simulating the Internet. *IEEE/ACM Transactions on Networking*, 9(4):392–403, August 2001.
- [31] Security Focus. Deepsight Threat Management Service, http://www.securityfocus.com/corporate/ products/tms.
- [32] Security Focus. MacOS X SoftwareUpdate Arbitrary Package Installation Vulnerability, http:// online.securityfocus.com/bid/5176.
- [33] Security Focus. www.securityfocus.com/bugtraq/.
- [34] Stephanie Forrest, Anil Somayaji, and David Ackley. Building Diverse Computer Systems. In *IEEE* 6th Workshop on Hot Topics in Operating Systems, 1997.
- [35] Reg Foulkes and John Morris. Fighting worms in a large corporate environment: A design for a network anti-worm solution. In *Virus Bulletin International Conference*, pages 56–66, New Orleans, 2002. Virus Bulletin Ltd.

- [36] The Animal Liberation Front. http://www. animalliberationfront.com.
- [37] The Earth Liberation Front. In defense of all life, http://www.earthliberationfront.com.
- [38] Fyodor. The Art of Port Scanning, http:// www.insecure.org/nmap/nmap_doc.html.
- [39] Gamespy. Gamespy arcade, http://www. gamespyarcade.com.
- [40] Google. http://www.google.com/.
- [41] Mike Hall and Kevin Wiley. Capacity verification for high speed network intrusion detection systems. In Andreas Wespi, Giovanni Vigna, and Luca Deri, editors, *RAID*, pages 239–251, Zurich, Switzerland, 2002. Springer-Verlag.
- [42] Shane Harris. Disease experts, medical supplies sent to New York. http://www.c-i-a.com/pr0302.htm, 2001.
- [43] Herbert HexXer. Codegreen beta release, http:// online.securityfocus.com/archive/82/211428.
- [44] Metacom Inc. Tivolicertified.com, http:// www.metacom.net/v2/tivoli/events/tam.asp.
- [45] Microsoft Inc. Security products and technologies center, http://www.microsoft.com/traincert/centers/ security.asp.
- [46] Symantec Inc. W32.gnuman.worm, http:// securityresponse.symantec.com/avcenter/venc/ data/w32.gnuman.worm.html.
- [47] Xilinx Inc. Virtex ii pro platform fpga complete data sheet, http://direct.xilinx.com/bvdocs/publications/ ds083.pdf.
- [48] itsecure. OpenSSH Trojan Horse, http://www. itsecure.com.au/alerts/alert.htm?alertid=95.
- [49] T. Jim, G. Morrisett, D. Grossman, M. Hicks, J. Cheney, and Y. Wang. Cyclone: A safe dialect of C. In USENIX Annual Technical Conference, Monterey, CA, June 2002.

- [50] Markus Kern. Re: Codegreen beta release, http:// online.securityfocus.com/archive/82/211462.
- [51] Darrell Kienzle. Private communication.
- [52] Eric Korpela, Dan Werthimer, David Anderson, Jeff Cobb, and Matt Lebofsky. Seti@home: Massive distributed computation for seti, http:// www.computer.org/cise/articles/seti.htm.
- [53] Kaspersky Labs. W95/CIH (a.k.a Chernobyl), http:// www.viruslist.com/eng/viruslist.html?id=3204.
- [54] Message Labs. W32/bugbear-ww, http://www. messagelabs.com/viruseye/report.asp?id=110.
- [55] Richard Lippmann, Robert K. Cunningham, David J. Fried, Isaac Graf, Kris R. Kendall, Seth E. Webster, and Marc A. Zissman. Results of the 1998 DARPA Offline Intrusion Detection Evaluation. In *Proceedings of Recent Advances in Intrusion* Detection (RAID), 1999.
- [56] Richard Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba, and Kumar Das. The 1999 DARPA off-line intrusion detection evaluation. *Computer Networks*, 34(4):579–595, October 2000.
- [57] John McHugh. Testing Intrusion detection systems: A critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. ACM Transactions on Information and System Security, 3(4):262–294, November 2000.
- [58] Brian McWilliams. Yaha Worm Takes out Pakistan Government's Site, http://online.securityfocus.com/ news/501.
- [59] Microsoft. ActiveX Controls, http://www.microsoft. com/com/tech/activex.asp.
- [60] Domas Mituzas. FreeBSD Scalper Worm, http:// www.dammit.lt/apache-worm/.
- [61] David Moore. Observing small or distant security events, http://www.caida.org/outreach/ presentations/2002/usenix_sec/.

- [62] David Moore, Colleen Shannon, and k claffy. Codered: a case study on the spread and victims of an Internet worm. In *The Second Internet Measurement Workshop*, pages 273–284, November 2002.
- [63] Carey Nachenberg. Personal communication, November 2002.
- [64] George Necula, Scott McPeak, and Westley Weimer. CCured: Type-Safe Retrofitting of Legacy Code. In Proceedings of the Principles of Programming Languages. ACM, 2002.
- [65] Netcraft. The Netcraft Survey, http://www.netcraft. com.
- [66] NORMAN. Norman Corporate History, http:// www.norman.no/about_norman.shtml.
- [67] Lab Tests Online. The Laboratory Response Network: Helping Laboratories to Prepare for Potential Terrorist Attacks, http://www.labtestsonline.org/lab/ labnetwork-4.html, May 2002.
- [68] Vern Paxson. Growth Trends in Wide-Area TCP Connections. *IEEE Network*, 8(4):8–17, July 1994.
- [69] Sam Phillips. dasbistro.com default.ida responder. http://sunsite.bilkent.edu.tr/pub/infosystems/% lphpweb/default.txt.
- [70] Rob Pike, Dave Presotto, Sean Dorward, Bob Flandrena, Ken Thompson, Howard Trickey, and Phil Winterbottom. Plan 9 from bell labs, http:// www.cs.bell-labs.com/sys/doc/9.pdf.
- [71] Honeynet Project. The Reverse Challenge, http:// project.honeynet.org/reverse.
- [72] OpenBSD project. Revised openssh security advisory, http://www.openssh.com/txt/preauth.adv.
- [73] The Honeynet Project. http://www.honeynet.org/.
- [74] The Honeynet Project. Know Your Enemy: Motives, http://project.honeynet.org/papers/motives/.
- [75] The Honeynet Project. The reverse challenge: http:// www.honeynet.org/reverse/.

- [76] Niels Provos. Honeyd: Network rhapsody for you, http://www.citi.umich.edu/u/provos/honeyd/.
- [77] Richard Rashid, Robert Baron, Alessandro Forin, David Golub, Michael Jones, Daniel Julin, Douglas Orr, and Richard Sanzi. Mach: A foundation for open systems. In Proceedings of the Second Workshop on Workstation Operating Systems(WWOS2), September 1999.
- [78] RedHat. Redhat secrity courses, http:// www.redhat.com/training/security/courses/.
- [79] Ryan Russell. Private communication.
- [80] Markus Schmall. Bulding Anna Kournikova: An Analysis of the VBSWG Worm Kit, http:// online.securityfocus.com/infocus/1287.
- [81] F secure Inc. Global slapper worm information center, http://www.f-secure.com/slapper/.
- [82] Gobbles Security. Remote Apache 1.3.x Exploit, http://online.securityfocus.com/archive/1/277830.
- [83] Alex Snoeren, Craig Partridge, Luis Sanchez, Christine Jones, Fabrice Tchakountio, Beverly Schwartz, Stephen Kent, and W.Timothy Strayer. Single-Packet IP Traceback. *IEEE/ACM Transactions on Networking*, 10(6), December 2002.
- [84] Colusa Software. Omniware: A Universal Substrate for Mobile Code, http://citeseer.nj.nec.com/ software950mniware.html, 1995.
- [85] Valve Software. Half life, http://www.half-life.com.
- [86] Stuart Staniford, Vern Paxson, and Nicholas Weaver. How to 0wn the Internet in Your Spare Time. In Proceedings of the 11th USENIX Security Symposium. USENIX, August 2002.
- [87] Symantec. Symantecs Corporate History, http:// www.symantec.com/corporate/index.html.
- [88] Symantec. W32.Benjamin.Worm, http:// securityresponse.symantec.com/avcenter/venc/ data/w32.benjamin.worm.html.

- [89] Symantec. W32.Sonic.worm, http:// securityresponse.symantec.com/avcenter/venc/ data/w32.sonic.worm.html.
- [90] Max Vision. Whitehats: Lion Internet Worm Analysis, http://www.whitehats.com/library/worms/lion/.
- [91] Max Vision. Whitehats: Ramen Internet Worm Analysis, http://www.whitehats.com/library/worms/ ramen/.
- [92] Inc VMware. VMware, http://www.vmware.com/.
- [93] David Wagner and Drew Dean. Intrusion detection via static analysis. In *IEEE Symposium on Security and Privacy*. IEEE, 2001.
- [94] David Wagner, Jeffrey S. Foster, Eric A. Brewer, and Alexander Aiken. A First Step towards Automated Detection of Buffer Overrun Vulnerabilities. In *Network and Distributed System Security Symposium*, pages 3–17, San Diego, CA, February 2000.
- [95] David Wagner and Paolo Soto. Mimicry Attacks on Host-Based Intrusion Detection. In ACM CCS, 2002.
- [96] 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.
- [97] World Health Organization. Communicable Disease Surveillance and Response, http://www.who.int/ emc/.