A Taxonomy of Computer Worms *

Nicholas Weaver† UC Berkeley

Vern Paxson‡ ICSI

Stuart Staniford§ Silicon Defense

Robert Cunningham¶ MIT Lincoln Laboratory

ABSTRACT

To understand the threat posed by computer worms, it is necessary to understand the classes of worms, the attackers who may employ them, and the potential payloads. This paper describes a preliminary taxonomy based on worm target discovery and selection strategies, worm carrier mechanisms, worm activation, possible payloads, and plausible attackers who would employ a worm.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Invasive Software*

General Terms

Security

Keywords

computer worms, mobile malicious code, taxonomy, attackers, motivation

*Portions of this work were performed under DARPA contract N66001-00-C-8045.
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.
Other Portions of this work were sponsored by DARPA under contract F19628-00-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by the United States Government.
†nweaver@cs.berkeley.edu
Portions of this work were completed as an employee of Silicon Defense.
‡vern@icir.org
Additional Support from NSF grant ITR-0205519
§stuart@silicondefense.com
¶rkc@ll.mit.edu

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

1. INTRODUCTION

A computer *worm* is a program that self-propagates across a network exploiting security or policy flaws in widely-used services. They are not a new phenomenon, having first gained widespread notice in 1988 [16].

We distinguish between worms and *viruses* in that the latter infect otherwise non-mobile files and therefore 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 Staniford et al [47] 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 hide their spread in otherwise unconnected user activity. 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 activation, as a type of worm.

In order to understand the worm threat, it is necessary to understand the various types of worms, payloads, and attackers. We attempt to construct a preliminary taxonomy of the various possible worms, payloads, and attackers as an initial guide to plausible defenses. This taxonomy is necessarily incomplete, simply because new tactics, payloads, and attackers may arise. Nevertheless, we believe this taxonomy is generally complete with regard to the current situation, including several strategies not yet seen in the wild.

This taxonomy is based on several factors: *target discovery*, *carrier*, *activation*, *payloads*, and *attackers*. Target discovery represents the mechanism by which a worm discovers new targets to infect (§2). The carrier is the mechanism the worm uses to transmit itself onto the target (§3). Activation is the mechanism by which the worm's code begins operating on the target (§4). Payloads are the various non-propagating routines a worm may use to accomplish the author's goal (§5). Finally, the various possible attackers have different motives and would therefore utilize different payloads (§6).

In addition, it is important to note that worms needn't be confined to a single type within each category. Some of the most successful worms are *multi-modal*, employing multiple means of target discovery, carrier, payload, etc., where the combination enables the worm to surpass defenses (no matter how effective) that address only a single type of worm.

In Appendix A, we also discuss the notion of the *ecology* within which worms exist, and which enables their successful spread.

2. TARGET DISCOVERY

For a worm to infect a machine, it must first discover that the machine exists. 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. If a defense blocks a given strategy, this can prevent an entire class of worms from propagating.

Scanning: Scanning entails probing a set of addresses to identify vulnerable hosts. Two simple forms of scanning are *sequential* (working through an address block using an ordered set of addresses) 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 [15, 6, 33] and worms which require timer or user based activation [29]. 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 [32], and therefore infected relatively few machines. Code Red I compromised about 360,000 machines [34], a small fraction of the estimated 10,000,000 machines running IIS [36], though the evidence indicates this may have been essentially the entire publicly-accessible population of vulnerable IIS machines [47].

There are several optimizations which apply to scanning worms. A highly effective optimization is a preference for local addresses [8, 6]. Although this may be slightly inferior for Internet-scale propagation [9], it enables the worm to exploit a single firewall breach to scan the entire local network. Permutation scanning [47] enables a worm to utilize distributed coordination to more effectively scan the net and to determine when the bulk of the network is infected.

The most effective optimization is a *bandwidth-limited* scanner. Many 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. The alternate, a bandwidth-limited scanner, is substantially faster. Slammer/Sapphire [33] was inadvertently a bandwidth-limited scanner as a side effect of its single packet UDP design, though a TCP-based bandwidth-limited scanner can also be constructed if the worm can craft raw TCP packets.

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 could have required 2 hours if it contained sophisticated scanning routines or targeted a more widespread vulnerability [47], or less than 15 minutes if it utilized a bandwidth-limited scanner [33].

Scanning is highly anomalous behavior, so devices can effectively detect scanning worms as being very different from normal traffic. Both the Williamson "virus throttle" [54, 51] and the Silicon Defense CounterMalice product [12] detect scanning related anomalies and respond by restricting traffic, representing defenses designed to stop an entire *family* of worms.

Pre-generated Target Lists: An attacker could obtain a target list in advance, creating a "hit-list" of probable victims [47]. 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 or open access points can be used to perform small-scale scans. 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. We have not yet detected a hitlist-based worm in the wild.

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, the Gamespy [23] service 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 [46] or others, even when the target population is relatively small, as there are metaservers which can be queried to discover the vulnerable machines. 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 risk is significant due to the great speed such a worm could achieve. One mitigating factor is that querying a metaserver is application-specific, so toolkits are less likely to reduce a worm author's required effort.

Internal Target Lists: Many applications contain information about other hosts providing vulnerable services. 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 [16] used topological techniques 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 can be incredibly fast.

In the presence of defenses, a worm could further expand the graph by communicating information known by one instance to other instances. Although this won't increase the

speed of a topological worm, it may be important for worms which are attempting to bypass defenses.

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 already known machines, the new victims are likely normal destinations for communication. This observation suggests that highly distributed sensors may be needed to detect topological worms.

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 during target discovery, which potentially makes them highly stealthy. *Contagion* [47] worms are passive worms which rely on normal communication to discover new victims.

There have been many passive worms, such as the Gnuman [24] bait worm and the CRClean [27] "anti-worm". 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, it spreads slowly.

Although never released, CRClean did not require human activation. This worm waits for a Code Red II related probe. When it detects an infection attempt, it responds by launching a counterattack. If this counterattack is successful, it removes Code Red II and installs itself on the machine. Thus CRClean spreads without any scanning.

2.1 Toolkit Potential

Some target discovery 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 [43] and application-dependent [43, 52] toolkits have been seen in the wild, and it is likely that such toolkits will become more widespread and sophisticated.

The application independent portions of a toolkit could 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.

Except for the exploit, scanning worms are not application-specific. Thus an attacker can add a new exploit to an existing worm framework or toolkit. The Slapper [45] worm was one such case, where the attacker inserted a new exploit into the Scalper [32] source code. This suggests that the window between when a vulnerability is published and when a scanning worm can be released is nearly zero if an attacker desires it, as the general scanning worm framework can be expressed as a toolkit. Similarly, a hitlist is also a generic structure which is constructed in advance, and would also work well in a worm toolkit.

Fortunately, topology is often application-specific, so each application requires its own toolkit for extracting it. Targeting that requires metaservers or passive analysis of communication is similarly application-specific. Thus, although toolkits can be developed, they are necessarily less generally applicable.

Toolkits already exist for email worms [43], providing common mechanisms for extracting email addresses, the topological information needed by these worms. These toolkits are effective because they aren't general topological-worm toolkits, but toolkits which attack a single, widely-deployed class of applications: email programs.

3. PROPAGATION CARRIERS AND DISTRIBUTION MECHANISMS

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 [27], also use self-carried propagation.

Second Channel: Some worms, such as Blaster [31], require a secondary communication channel to complete the infection. Although the exploit uses RPC, the victim machine connects back to the infecting machine using TFTP to download the worm body, completing the infection process.

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 [47] is an example of a passive worm that uses embedded propagation.

An embedded strategy, although relatively 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, as is how far from the natural patterns of communication such a worm could deviate in order to hasten its propagation without compromising its stealthiness.

Likewise, the distribution of the worm body or related payloads can either be one-to-many, as when a single site provides a worm or module to other sites once they've been initially infected; many-to-many, as when multiple copies propagate the malicious code; or a hybrid approach where the basic worm propagates in a many-to-many manner with updates received through a central site. In general, many-to-many distribution can be considerably faster, if a limiting factor is the time it takes to perform the distribution. Many-to-many distribution also removes the ability for others to block further distribution by removing the source of the malicious code from the Internet.

4. ACTIVATION

The means by which a worm is activated on a host also drastically affects how rapidly a worm can spread, because some worms can arrange to be activated nearly immediately whereas others may wait days or weeks 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 email-worm [3] indicate urgency on the part of someone you know ("Attached is an important message for you"); others, such as the Iloveyou [4] attack, appeal to individuals' vanity ("Open this message to see who loves you"); and others, such as the Benjamin [49] 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 [19]). The continued spread of these worms is disturbing, as they can be effectively used as secondary vectors[1] such as was the case for Nimda [6], and/or to install additional malicious software such as programs which allow an attacker to control a machine [18].

Human Activity-Based Activation: Similarly, many worms are activated when the user performs some activity not normally related to a worm, such as resetting the machine, logging in and therefore executing login scripts, or opening a remotely infected file. This activation mechanism is commonly seen in *open shares* windows worms (such as one of Nimda's secondary propagation techniques [6]) which will begin execution on the target machine either when the machine is reset or the user logs in, as these worms write data to the target disk without being able to directly trigger execution.

Scheduled Process Activation: The next fastest worms activate using scheduled system processes. Such programs can propagate through mirror sites (e.g., OpenSSH Trojan [25]), 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 [20] to infect the target. Other systems periodically 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 may 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 [15] exploiting IIS Web servers) or in the libraries that the services use (e.g., XDR [7]). 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.

[1] A 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.

5. PAYLOADS

The payload, the code carried by the worm apart from the propagation routines, is limited only by the goals and imagination of the attacker. 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 and machine load (as seen with both the Morris worm [16] and Slammer [33]) and by actively advertising vulnerable machines.

Internet Remote Control: Code Red II opened a trivial-to-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 [39] which exploited the backdoor to issue the commands to disable IIS and reset the machine.

Spam-Relays: Part of the Sobig worm's associated trogan [48], creates an open-mail relay for use by spammers. By creating numerous relay machines across the Internet, spammers can avoid blackhole-based mechanisms which block known-spamming IP addresses.

HTML-Proxies: Another aspect of Sobig's trojan is the distribution of web-proxies. By redirecting web requests (through DNS) to randomly selected proxy machines, it becomes significantly more difficult for responders to shut down compromised websites which are used for various illegal activities, including scams which attempt to entise users to input financial data (a technique called *phishing*).

Internet DOS: Another common payload is a Denial of Service (DOS) attack. Code Red [15], Yaha [30], and others have all contained DOS tools, either targeted at specific sites or retargetable under the attacker's control. Distributed DOS (DDOS) tools such as Stacheldraht [13] 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, and response channels all at the same time.

Data Collection: Computers are increasingly used to store and manipulate sensitive data. A worm could use target these capabilities, and some already have. SirCam [5] 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.

Criminals are sometimes interested in *identity theft*, and significant subsets of the blackhat community are involved

in harvesting credit cards [2] and could use worms to search for this information. After discovery, the results could be encrypted and transmitted through various channels.

Access for Sale: An extension on remote control and data-collection payloads is access for sale [42]. With this payload, the worm will allow remote access to paying customers, but only to the specific victims to which the customer desires access.

Data Damage: There have been many viruses and email worms, such as Chernobyl [28] or Klez [19], which contained time-delayed data erasers. Since worms can propagate much faster, they could start erasing or manipulating data immediately after infecting a system. Data could also be encrypted instead of destroyed as part of an extortion scheme [55]. In addition, a worm could distribute sensitive information to cause general confusion.

Physical-world Remote Control: In addition to altering the attacked computer and network, attacks can affect non-Internet objects, services and expectations. Networked computers are used to control physical-world objects, often through supervisory control and data acquisition (SCADA) systems, and a worm could target those computers.

Computers can also be used to influence the actions of humans. A coercive payload might do 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.

Physical-world DOS: In addition, computers can deny service in the physical world. For example, a worm can use attached modems to dial emergency services such as *911* [44] or other telephone targets, or use catalog-registration features to saturate the physical mailboxes of a large number of targets [1].

Physical-world Reconnaissance: As an example of this type of attack, a computer worm could "wardial" via an attached modem[2] to conduct further reconnaissance for later, non-Internet based attacks.

Physical-world Damage: The most direct object to damage is the infected computer. Although the diversity of BIOSs prevents a general reflashing, it would be possible for a worm to include reflashing routines for several common BIOSs, using the same mechanisms employed by the Chernobyl virus [28]. Since the FLASH ROMs are often soldered to the motherboard, such an attack could effectively destroy particular motherboards when there doesn't exist a protected recovery BIOS or similar mechanisms.

Worm Maintenance: The final class of payload is one that is used to maintain the worm. Past worms such as W32/sonic [50] have included a crude update mechanism: querying web sites for new code. W32/hybris [18] also checked Usenet newsgroups and cryptographically verified the modules before execution. Similarly, DDoS tools have also enabled updates to the zombie program [14]. 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.

[2]Wardialing is the process of scanning telephone numbers for answering modems.

6. 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.

Experimental Curiosity: Although the technology is well understood, there is a continual tendency for various individuals to experiment with viruses and worms. The Morris worm was not only a pioneering event, but an experiment which escaped. Likewise, the ILoveYou [4] worm was designed by a student and proposed as a thesis project before it was released.

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 [40]. The people who do this are typically unorganized individuals or small groups 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 wares. International companies or organized crime members could participate in this type of attack, and the targets range from specific companies to economic infrastructure.

Extortion and Criminal Gain: Another potential profit motive is extortion or other criminal gain. Since a well-constructed worm could launch an effective 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. Likewise, a worm which searches for credit-card information would represent another criminally-motivated payload.

Likewise, the Sobig worm's trojan has been linked to several semi-legal and illegal activities, as it creates open mail relays and web proxes which are used for spamming, serving of pornographic material, and phishing.

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 [30] 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 attractive economic weapons for those who believe that large corporations are an evil, as well as those with animosity directed against particular nations or governments. Or, if desired, the attack could target all infectible computers, in an attempt to cause the maximum damage.

Such an attack would be *economic* terrorism, where the goal is to cause significant monetary disruption, not loss of life. Attackers could include Al-Quaeda [10], splinter groups derived from the antiglobalization movement, or ecoterrorist groups such as ELF [22] or ALF [21], which claim to exclusively practice economic terrorism.

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

7. CONCLUSION

We have developed a taxonomy of worms, based on target discovery, carrier, activation, payload, and attackers. The carrier, activation, and payload are independent of each other, and describe the worm itself. Those who want to help develop more robust defenses can focus on preventing worms that use one or more of the techniques described here. We also include a section on attackers and their motivations, because worms are ultimately written by humans, and sometimes the easiest way to defend against a worm is to remove the motivation for writing a worm in the first place.

APPENDIX

A. 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 [47], to perform audits of widespread applications to determine vulnerabilities and resistance to worm based attacks.

Buffer Overflows: One of the largest sources of vulnerabilities is the continued use of the C and C++ languages, which allows buffer overflow attacks and related code-injection 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 and ProPolice [11, 17], Software Fault Isolation [53], unexecutable stacks and heaps [38, 37], and "Safe C" dialects like CCured [35] and Cyclone [26]. Yet none of these have been widely adopted.

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 malicious code authors in numerous contexts.

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 [32], 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.

Patch Deployment: It has been commonly observed that patches are often undeployed [41]. Partially it is neglect, but another concern is simply the risk involved: patches need to be tested within a particular institution before deployment. Thus many installations will queue up multiple patches, test them as a group, and then install all the patches simultaneously.

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.

B. REFERENCES

[1] Simon Byers, Aviel Rubin, and David Kormann. Defending against internet-based attack on the physical world, http://www.avirubin.com/scripted.attacks.pdf.

[2] Cardcops. http://www.cardcops.com.

[3] CERT. CERT Advisory CA-1999-04 Melissa Macro Virus, http://www.cert.org/advisories/ca-1999-04.html.

[4] CERT. CERT Advisory CA-2000-04 Love Letter Worm, http://www.cert.org/advisories/ca-2000-04.html.

[5] CERT. CERT Advisory CA-2001-22 w32/Sircam Malicious Code, http://www.cert.org/advisories/ca-2001-22.html.

[6] CERT. CERT Advisory CA-2001-26 Nimda Worm, http://www.cert.org/advisories/ca-2001-26.html.

[7] CERT. CERT Advisory CA-2002-25 Integer Overflow in XDR Library, http://www.cert.org/advisories/ca-2002-25.html.

[8] CERT. Code Red II: Another Worm Exploting Buffer Overflow in IIS Indexing Service DLL, http://www.cert.org/incident_notes/in-2001-09.html.

[9] Zesheng Chen, Lixin Gao, and Kevin Kwiat. Modeling the spread of active worms. In *IEEE INFOCOM 2003*. IEEE, April 2003.

[10] ComputerWorld. Al-qaeda poses threat to net, http://www.computerworld.com/securitytopics/security/story/0,10801,76150,00.html.

[11] 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.

[12] Silicon Defense. Countermalice worm containment, http://www.silicondefense.com/products/countermalice/.

[13] David Dittrich. The Stacheldraht Distributed Denial of Service Attack Tool, http://staff.washington.edu/dittrich/misc/stacheldraht.analysis.

[14] David Dittrich. The Tribe Flood Network Distributed Denial of Service Attack Tool, http://staff.washington.edu/dittrich/misc/tfn.analysis.

[15] eEye Digital Security. .ida "Code Red" Worm, http://www.eeye.com/html/research/advisories/al20010717.html.

[16] Mark Eichin and Jon Rochlis. With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988. In *IEEE Computer Society Symposium on Security and Privacy*, 1989.

[17] Hiroaki Etoh. Gcc extentions for protecting applications from stack-smashing attacks, http://www.research.ibm.com/trl/projects/security/ssp/.

[18] F-Secure. F-Secure Computer Virus Information Pages: Hybris, http://www.f-secure.com/v-descs/hybris.shtml.

[19] Peter Ferrie. W32//Klez, http://toronto.virusbtn.com/magazine/archives/200207/klez.xml.

[20] Security Focus. MacOS X SoftwareUpdate Arbitrary Package Installation Vulnerability, http://online.securityfocus.com/bid/5176.

[21] The Animal Liberation Front. http://www.animalliberationfront.com.

[22] The Earth Liberation Front. In defense of all life, http://www.earthliberationfront.com.

[23] Gamespy. Gamespy arcade, http://www.gamespyarcade.com.

[24] Symantec Inc. W32.gnuman.worm, http://securityresponse.symantec.com/avcenter/venc/data/w32.gnuman.worm.html.

[25] itsecure. OpenSSH Trojan Horse, http://www.itsecure.com.au/alerts/alert.htm?alertid=95.

[26] 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.

[27] Markus Kern. Re: Codegreen beta release, http://online.securityfocus.com/archive/82/211462.

[28] Kaspersky Labs. W95/CIH (a.k.a Chernobyl), http://www.viruslist.com/eng/viruslist.html?id=3204.

[29] Message Labs. W32/bugbear-ww, http://www.messagelabs.com/viruseye/report.asp?id=110.

[30] Brian McWilliams. Yaha Worm Takes out Pakistan Government's Site, http://online.securityfocus.com/news/501.

[31] Jason V Miller, Jesse Gough, Bartek Kostanecki, Josh Talbot, and Jensenne Roculan. Microsoft dcom rpc worm alert, https://tms.symantec.com/members/analystreports/030811-alert-dcomworm.pdf.

[32] Domas Mituzas. FreeBSD Scalper Worm, http://www.dammit.lt/apache-worm/.

[33] David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, and Nicholas Weaver. Inside the slammer worm. *IEEE Magazine of Security and Privacy*, pages 33–39, July/August 2003 2003.

[34] David Moore, Colleen Shannon, and k claffy. Code-red: a case study on the spread and victims of an Internet worm. In *The Second Internet Measurement Workshop*, pages 273–284, November 2002.

[35] George Necula, Scott McPeak, and Westley Weimer. CCured: Type-Safe Retrofitting of Legacy Code. In *Proceedings of the Principles of Programming Languages*. ACM, 2002.

[36] Netcraft. The Netcraft Survey, http://www.netcraft.com.

[37] Openbsd 3.3, http://www.openbsd.org/33.html.

[38] The homepage of the pax team, http://pageexec.virtualave.net/.

[39] Sam Phillips. dasbistro.com default.ida responder. http://sunsite.bilkent.edu.tr/pub/infosystems/%lphpweb/default.txt.

[40] The Honeynet Project. Know Your Enemy: Motives, http://project.honeynet.org/papers/motives/.

[41] Eric Rescorla. Security holes ... who cares? In *Proceedings of the 12th USENIX Security Symposium*, pages 75–90. USENIX, August 2003.

[42] Stuart Schechter and Michael Smith. Access for sale: A new class of worm. In *First Workshop on Rapid Malcode WORM*, October 2003.

[43] Markus Schmall. Bulding Anna Kournikova: An Analysis of the VBSWG Worm Kit, http://online.securityfocus.com/infocus/1287.

[44] McAffe Secuirty. W95/firkin.worm, http://vil.mcafee.com/dispvirus.asp?virus_k=98557.

[45] F secure Inc. Global slapper worm information center,

http://www.f-secure.com/slapper/.

[46] Valve Software. Half life, http://www.half-life.com.

[47] 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.

[48] Joe Stewart. Sobig.e: Evolution of the worm. http://www.lurhq.com/sobig-e.html.

[49] Symantec. W32.Benjamin.Worm, http://securityresponse.symantec.com/avcenter/venc/data/w32.benjamin.worm.html.

[50] Symantec. W32.Sonic.worm, http://securityresponse.symantec.com/avcenter/venc/data/w32.sonic.worm.html.

[51] Jamie Twycross and Matthew M Williamson. Implementing and testing a virus throttle. In *Proceedings of the 12th USENIX Security Symposium*, pages 285–294. USENIX, August 2003.

[52] Max Vision. Whitehats: Ramen Internet Worm Analysis, http://www.whitehats.com/library/worms/ramen/.

[53] 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.

[54] Matthew M Williamson. Throttling viruses: Restricting propigation to defeat mobil malicious code. In *Annual Computer Security Applications Conference*, 2002.

[55] Adam Young and Moti Yung. Cryptovirology: Extortion based security threats and countermeasures. In *IEEE Symposium on Security and Privacy*, pages 129–141, Oakland, CA, 1996. IEEE Computer Society Press.