Digging For Worms, Fishing For Answers

The CERIAS Intrusion Detection Research Group*
Center for Education and Research in Information Assurance and Security
Purdue University
West Lafayette, IN

Abstract

Worms continue to be a leading security threat on the Internet. This paper analyzes several of the more widespread worms and develops a general life-cycle for them. The life-cycle, from the point of view of the victim host, consists of four stages: target selection, exploitation, infection, and propagation. While not all worms fall into this framework perfectly, by understanding them in this way, it becomes apparent that the majority of detection techniques used today focus on the first three stages. This paper presents a technique that is used in the fourth stage to detect the class of worms that use a horizontal scan to propagate. An argument is also made that detection in the fourth stage is a viable, but under-used technique.

1. Introduction

Worms continue to be a serious threat to the Internet as a whole. A worm is an "Independent program that replicates from machine to machine across network connections often clogging networks and information systems as it spreads" [19]. To date, the worms that have become widespread have been relatively benign. However, as worm writers become more sophisticated, the potential damage caused by worms becomes incalculable. Warhol and Flash worms [18] illustrate ways in which worms can become more sophisticated and rapidly spread. In fact, more recent worms such as Slapper [4] have set up Distributed Denial of Service (DDoS) networks as they spread.

To model how worms behave and propagate, a four stage life-cycle was developed. Our life-cycle is constructed from the point of view of an uninfected host. First, the host is selected as a target by the worm from a remote machine. The host is then compromised through some sort of exploit. After being compromised, the worm then infects the host. Finally, the worm propagates by choosing other targets to infect. Not all worms fit perfectly within this framework; however, from a conceptual standpoint, the actions in each stage are present in a worm's behavior and can be successfully applied to most of the worms in the past few years.

As worms have been identified as a significant security threat, a number of techniques have been developed to defend against them. However, the majority of the focus has been on the first three stages of a worm's life-cycle. For example, signature based network intrusion detection systems are used to detect exploits and host based intrusion detection systems are used to detect infections based upon modified, deleted, or added files.

This paper focuses on techniques that can be used to detect and contain worms in the fourth stage, propagation. A class of worms that behave similarly in the fourth stage is also identified. By watching outbound network traffic from an infected host, it is possible to detect these worms as they attempt to propagate. There are a number of advantages in this form of detection, including being notified of an infected host and containing any further infection on the local network or in other domains (potentially protecting one's reputation). Furthermore, even if a worm is not detected when it enters the network, it may be possible to detect it on its way out.

The next section describes the four stage life-cycle of a worm in greater detail. Section 3 discusses the techniques currently being used to detect worms in the context of the four stage life-cycle. Section 4 looks at the different techniques that can be used to detect a worm in the fourth stage. Finally, a preliminary implementation, future work and conclusions are presented.

2 The Four Stage Life-cycle

By analyzing the behavior of several worms, it becomes apparent that they exhibit many similar activities. By grouping their common actions, a four stage life-cycle emerges that characterizes worms' behavior.

*Florian Buchholz, Thomas E. Daniels, James P. Early, Rajeev Gopalakrishna, R. Patrick Gorman, Bejamin A. Kuperman, Sofie Nystrom, Addam Schroll, and Andrew Smith. This paper was partially funded by the sponsors of CERIAS

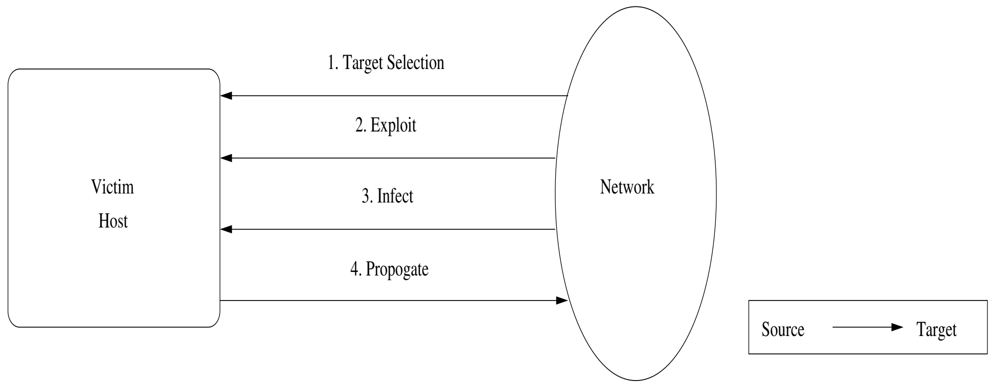

Figure 1. The Four Stage Life-cycle of a Worm

The four stage life-cycle, as shown in Figure 1, is from the point of view of a host that is about to become infected. The arrows represent the source of each stage, but are not necessarily indicative of separate network traffic or connections.

The four stages in the life-cycle are:

1. Target selection
2. Exploitation
3. Infection
4. Propagation

In practice, the actions in the first and fourth stages are the same. However, from the point of view of the attacked host, they appear differently. This will become clear as each of these stages is described in more detail. The Lion worm, described in both [2] and [5], will be used as an example to illustrate the actual actions of a worm in each stage.

2.1 Target Selection

The target selection stage is the phase when an uninfected host is chosen to be attacked. This is where the worm performs reconnaissance to determine other potential victims.

From the point of view of the host that is about to be infected, this may be as simple as a single network probe. For example, the Lion worm simply probes a potential victim to see if it is running a service on port 53. If there is indeed a service running, then the worm moves into the second stage to attempt to exploit the victim.

2.2 Exploitation

The exploitation phase is when the worm compromises the target by exploiting a particular vulnerability. Oftentimes, worms use well known vulnerabilities and published exploits to compromise their target.

In the Lion example, a BIND exploit [1] is used to compromise the host and obtain root privileges. If the victim is successfully compromised, then the worm moves into the third stage, infection.

2.3 Infection

The infection stage is the most broad in the life-cycle, as the worm copies itself on to the victim machine and then performs any number of different actions. The line between the infection and exploitation stage is sometimes blurred. For example, the Code Red worm [3] actually loads itself into the victim's memory as part of the exploit used to compromise the victim. The infection stage is understood to be the time when the worm "sets up shop" on the newly infected machine. For example, the worm can open backdoors, change system files, or attempt to hide its presence by replacing system utilities with trojan horses.

The Lion worm connects to a predetermined website to download itself to the newly infected host, mails several system files (including `/etc/shadow`) to a predetermined email address, and adds an entry to `/etc/inetd.conf` to allow attackers to connect to port 10008 with root access.

After the worm has set itself up on the newly infected host, it then moves into the fourth stage of the life-cycle, propagation.

2.4 Propagation

In the propagation stage, the worm attempts to spread by choosing new targets. The difference between this stage and *target selection* is simply the point of view from which the actions take place. In target selection, a remotely infected host chooses the local host as a target, often in the form of a probe coming in through inbound network traffic. In the propagation stage, the infected local host is the one choosing a new target, using probes going out in outbound network traffic. This is an important distinction and allows new techniques to be used in worm detection.

The Lion worm begins its propagation by randomly choosing an IP address as a base. Then, starting with this base address, it uses a `TCP SYN` scan to probe on port 53. It repeatedly increments the address by one to get the next target it will probe. It scans well over a hundred hosts per second.

2.5 Four Stage Life-cycle Summary

Looking at the entire life-cycle for the Lion worm, a victim host would see:

- Stage One - A connection attempt on port 53
- Stage Two - A BIND exploit on port 53
- Stage Three - An outbound connection to a website to download the worm, filesystem changes to open a backdoor, and outbound email transferring sensitive system files
- Stage Four - A series of rapid outbound scans probing port 53 on randomly selected hosts

In Table 1, included at the end of the paper, several of the more prolific worms are broken into the described four stage life-cycle. Every worm in the table scans to find potential victims. The *Network Probe* category illustrates this, and lists the port numbers of any scanned ports. In addition, some worms use email in order to spread themselves. This category was included for completeness, but is not otherwise addressed. The *Exploit* category gives the targeted service and the CERT Advisory that describes the vulnerability. The *Infection* category includes columns for filesystem modifications, backdoors, and other activity. A filesystem modification includes everything from a copy of the worm being placed on the system to changing system files. A backdoor includes techniques such as adding new users to the system and opening a root shell server (in which case the port is listed). The *Other* column contains activities that do not fall into the first two, such as emailing out system files, connecting to a server to download the worm, or setting up a DDoS client. The *Propagation* category includes an *Other* column, which accounts for different spreading techniques such as a web server infecting a client.

This table gives the intuition behind how each worm can be detected or prevented in its different stages.

3 Detection Techniques in the First Three Stages

Having described the four stage life-cycle of a worm, it becomes apparent that the majority of tools used to combat worms focus on the first three stages. This section will look at the different techniques being used today in each stage, as well as the limitations that these tools face.

3.1 Detecting Target Selection

As discussed, from the point of view of the victim host, the target selection phase typically involves an inbound network probe on a particular port. Firewalls are designed to block this type of probe. If a remote machine is attempting to access a service it does not have permission to access, the firewall can block it. However, worms often target services that are available to the public, such as a web server. In this case, detection in this phase can be reduced to the detection of an inbound portscan.

The standard technique used to detect a port scan is "testing for X events of interest across a Y-sized time window [11]." For example, one can test for the same source address accessing 10 different ports on the same destination machine within a two second interval. This claim is supported by a survey of a number of different network intrusion detection systems that use slight variations of the aforementioned technique to detect portscans [17].

This technique is viable if the scanning host actually triggers X events; however, as noted earlier, the scanning host may only send one probe into a network. It does not need to scan multiple ports, or multiple hosts.

In the case where a target can be selected with one probe, such as a connection to a web server on port 80, very little can be done to detect the worm at this stage in the life-cycle.

3.2 Detecting Exploits

Because of the difficulty of detecting the scans described in the prior stage, much of the work that has been done to detect and prevent worms falls into detecting attempts to exploit vulnerabilities.

The first, and most effective line of defense against worms and any other network attack is to keep servers up-to-date with the latest patches. Many of the worms in circulation today exploit well known vulnerabilities for which patches exist (and existed even before the worms began spreading). Furthermore, many worms simply copy their predecessors and exploit the same vulnerabilities. Using Table 1, it can be seen that Code Red v1 and Code Red II both exploit the same vulnerability. Similarly, Adore exploits the same vulnerabilities that Lion and Ramen exploit.

Detecting exploits as they cross the network is the main goal of a Network Intrusion Detection System (NIDS). While many worms attack well known vulnerabilities, they also use well known exploits for which signatures have been written. In these cases, an up-to-date ruleset on a NIDS will generate an alert when a worm attempts to attack a host on the network.

The limitations of network intrusion detection systems and patches are well known. If a new vulnerability is attacked, and the exploit has never been seen in the wild, patches will probably not have been released and signatures will not have been written to detect it. Consequently, a host will be vulnerable, and a signature based NIDS will miss the exploit. Furthermore, there are several techniques that can be used to avoid detection by a NIDS, with varying degrees of success depending upon the actual NIDS that is deployed. These include techniques such as slightly varying the signature of the worm, overlapping fragmentation attacks, and others [14].

3.3 Detecting Infection

Due to the number of different actions a worm can take in this stage, there are many different techniques that may be used to detect infection. A few of them will be discussed here.

Oftentimes, once a worm has been captured on the Internet, antivirus products are quickly updated to detect it. Thus, if a particular file is suspected of being a worm, an up-to-date antivirus product may be able to detect it.

If a worm modifies important system files, such as replacing utilities with trojan horses or adding services in `/etc/inetd.conf`, a file integrity program such as Tripwire [9] can be used to detect these changes. Every worm in Table 1 makes a filesystem change of some kind. If monitored system files are changed, a tool like Tripwire can easily detect it.

If a worm starts a server on particular port, perhaps to open a backdoor into the system, combining the use of a network scanner [7] and system utilities such as `netstat` can detect it. For example, these tools could detect the root-shell servers set up by both the Sadmind and Lion worms.

Because of the number of different actions a worm can take, there is no single answer to detection in this stage. Instead, a number of different techniques may be of use that depend upon the particular worm. At the same time, it is easy to envision a worm with little noticeable footprint on the host system. For example, the worm could load itself into memory, and then remove any trace of its presence on the filesystem. In this case, detection in this stage becomes difficult.

Using the detection techniques from the first three stages of the life-cycle, administrators have a variety of tools that can protect their networks. However, if a worm were released that carefully targeted its victims, exploited a new vulnerability, and left a small footprint on the system, more would need to be done to detect its presence. That detection can take place in the fourth stage of the life-cycle.

4 Detecting Propagation

As described, a new, properly designed worm may evade detection in the first three stages. However, it can still be possible to detect its presence when it attempts to propagate. As much of the focus in the security community is in protecting the network from attack, a motivation for the importance of tools that can be used after a successful attack will also be discussed.

All the worms analyzed in this paper have one characteristic in common: they rapidly scan randomly selected IP addresses to find new victims. Thus, with this type of worm, detection reduces to a type of outbound scan detection.

4.1 Prior Work In Detecting Scans

GrIDS [16] is an intrusion detection system designed to detect large scale attacks on a network. It does so by building activity graphs of network traffic and analyzing them in relation to graphs that model different attacks. GrIDS has been used to detect the spread of a worm by looking for a tree-like connection graph resulting from the worm as it branches out to newly infected hosts. It can also detect sweeps, which occur when a host connects to many other hosts in succession.

Other intrusion detection systems can have outbound scan detection built in by using modules or the rule description language. The Bro IDS's policy scripting language can be used to detect scans. Essentially, it can be told to watch for the number of connections to cross a particular threshold [12], where a connection consists of a single source address and either a destination address or port. No timing mechanism is used.

A Snort Portscan preprocessor[1] [15] has also been developed. The preprocessor allows the specification of the network to monitor for portscans, the threshold number of ports that are accessed, the period of time the threshold must be exceeded in, and a log file for the results. A limitation of this preprocessor is that only the scan's destination network can be specified, thus it cannot be configured to *only* watch outbound scans. That is, one cannot specify a range of addresses to watch for originating scans.

While the tools and techniques exist to detect scan activity, the main focus has been on detecting inbound scans. At the same time, the goal has been to detect general port scan activity, as opposed to focusing solely on the more specialized type of scanning used by worms. While existing IDSs allow outbound scan detection to be built in, the prevalence of worms in the past few years suggests that this is not being done. Thus, the next section will narrow the classic X events in Y period of time technique to detect the outbound scans that are frequently used by worms.

[1] Written by Patrick Mullen

4.2 Revisiting the X Events in Y Period of Time Technique

Outbound scan detection has advantages over its counterpart – inbound scan detection. One such advantage is the ability to see all of the scans leaving the scanning host. As discussed in Section 3.1, the most widely used technique to detect a scan is to look for a certain number of events that occur within a set period of time. This technique is limited when watching inbound traffic, as a scan may only probe one host on any given network, and thus consist of only one event. However, watching outbound traffic may trigger enough events to detect the worm, even if it probes several different networks.

Many types of scans exist [6], and any detection mechanism would have to take this into account. Instead of focusing on particular scan types, the TCP SYN scan with a *horizontal scan footprint* [17] will be examined.

A horizontal scan occurs when a worm is only probing a particular port on a set of different hosts. The Lion worm uses a horizontal scan to probe port 53 on randomly selected IP addresses. In this case, the source IP address (`src_ip`) and the destination port (`dst_prt`) are the same across every probe, while the destination IP address (`dst_ip`) and the (`src_prt`) will change.

The worms in Table 1 represent a class of worms that use a horizontal scan to find potential victims. Thus, a technique that can detect a horizontal scan would also detect all of the worms in this table.

It will be assumed that detection is not performed on the infected host, but rather at a network sensor that can see all of the traffic generated by the infected host. At the same time, it should be noted that where a sensor should be placed on the network has been widely debated in the security community and is not always as straight-forward as it might first appear.

To detect this type of scan, the following C-like pseudocode can be used:

```
typedef (src_ip, dst_prt) conn;

/* Called for every outbound packet */
process_conn(conn) {
 if( conn.exists ) { /* We have seen */
   conn.counter++;
   if( conn.counter > THRESHOLD) {
     generate_alert();
   }
 }
 else if(src_ip in specified range) {

   /* Create a new pair */
   add(conn);
   conn.counter = 1;

   /* Period of time to count events */
   conn.timer   = QUANTUM;
 }
}

/* Called every second */
timer() {
  foreach( conn ) {
    conn.timer--;
    if( conn.timer == 0 )
      remove(conn);
  }
}
```

A key to this approach is determining the appropriate QUANTUM and THRESHOLD values. Or, determining what window should be used such that if we observe THRESHOLD repeated (`src_ip, dst_prt`) pairs, an alert is generated. These numbers will vary based on the characteristics of a particular network's traffic, and will need to be experimented with to find effective values.

While this technique is similar to what has been used in the past, the fundamental difference is that it is designed to watch *outbound* traffic for horizontal scans. It does this by making sure the source IP address originates from within the local network (the specified range). This technique is a viable one for worms that are designed to spread rapidly, such as the worms in Table 1 and Warhol worms [18].

This type of monitoring – watching outbound traffic to detect a worm after it has already infected the network – has additional advantages over standard inbound traffic monitoring, several of which are enumerated in [8]. A few of these advantages will be discussed next.

4.3 Increased Confidence in Alerts

Portscans in general are highly prevalent on the Internet, and though often used as a first step in an attack, by themselves they pose little threat. They are so common that administrators often do not have time to follow up on them, and thus consider them part of the background noise of the Internet.

While detection of an inbound portscan may be acceptable, a local host scanning out to the Internet is likely to violate a network's usage policy. In addition, detected outbound portscans should be much less frequent.

Pulling this together, an alert generated by an outbound portscan should carry more importance than an inbound scan. Thus, an administrator will have more incentive to investigate such alerts, and may discover a misbehaving user or a worm.

4.4 Containing the Damage

A potentially devastating characteristic of a worm is its ability to spread quickly across the Internet. Once a worm has been designed to automatically compromise a machine and then spread, authors can incorporate any other behavior they desire. Thus, the ability to contain a worm is important.

One tool that has been written to slow down worms and portscans is *LaBrea* [10]. Essentially, LaBrea allows a TCP connection to be opened to a nonexistent host, and then simply sits on the connection. Thus, when a worm attempts to connect to nonexistent IP addresses on one's network, it is tricked into believing that a machine is indeed there.

By watching outbound traffic, an administrator should be alerted when it appears that a worm is attempting to propagate from an infected local host. This gives an administrator the opportunity to go to the host, investigate, and take corrective action. Furthermore, as many worms target the local network in addition to remote machines, by detecting the worm's presence, the machines on the local network may be protected.

By containing the damage a worm does, and limiting its ability to spread, there is also the benefit of protecting one's public relations.

4.5 Public Relations

The widespread discovery that one has been infected with a worm can be a public relations nightmare. The reluctance to admit to suffering a security breach is well known. A recent study in [13] shows that in 2001, only 36% of those who were seriously attacked reported it to law enforcement. This number is actually up from a low of 17% in 1998. While there are a number of reasons for this low number, one reason is the potential damage to one's reputation. This is supported by the web pages that were set up listing networks that were infected during the height of the Code Red worm's spread. A number of these networks pulled themselves off-line to contain the worm and protect their image.

Once a worm's scanning behavior or exploit signature has been determined, administrators can easily tell when their network is being scanned by a worm and where the worm is coming from. Furthermore, once the scanning host is identified, it is easy to determine the entity (company, University, etc.) that is infected by a particular worm.

If the public finds out that an Internet security company, financial institution, or retail company has been infected by a worm, public confidence may be shaken. It can lead to questions such as: "If this security company cannot defend itself against a worm, how can it defend me?" or "If this bank has been infected by a worm, are my personal files and accounts safe?" Thus, it makes sense to do everything possible to contain a worm once it has infected the network.

5 Limitations

The technique described in this paper is a step in the right direction, but it does have limitations.

Just as one can envision a worm that evades detection in the first three stages, it is easy to envision one that evades detection in the fourth stage as well. For example, if the worm is content to spread slowly, an outbound scan will not be detected. Despite this, attempted detection in this stage succeeds in raising the bar in some way. Furthermore, a slow moving worm may work against its ultimate goals, as administrators will have more time to detect and remove it while notifying the general public. To date, the worms discovered in the wild have attempted to spread as quickly as possible, but there is no guarantee that future evolutions will do the same.

While the described technique attempts to detect outbound scans, worms may evolve to employ other techniques to spread in the future. Such examples may include embedding themselves into standard client/server `HTTP` traffic or taking advantage of the prevalent *peer-to-peer* networks [18].

6 Current Implementation and Future Work

Thus far, experimentation is still in the early stages. A prototype is in the development stage as a Snort preprocessor [15], based on the pseudo code in Section 4.2. Developing it as a preprocessor allows it run anywhere that Snort can run. It is currently designed to watch for TCP SYN scans, but will be augmented to handle other scans as described in [6].

While preliminary tests on an isolated network can detect rapid horizontal scans, the next step is to deploy it on a production network. This will allow it to be fine-tuned to determine if THRESHOLD and QUANTUM values can be found that are capable of detecting the propagation stage of a worm, while not generating excessive false positives.

Techniques to detect other types of outbound scans are also going to be investigated. One such scan is the *vertical scan*. This occurs when a scanning host probes several ports on the same target. For example, the default setting in Nmap can be used to determine the services a host is running between ports 1 and 1024.

Another area of investigation will be in coordinating different detection mechanisms in the four stages of the lifecycle. For example, it may be possible to correlate inbound traffic that may be indicative of a worm with outbound traffic that looks like a worm attempting to propagate. If both of these observations occur together, there is a higher likelihood of an infection than if just one observation were made alone.

7 Conclusions

The majority of the effort being put into stopping worms today focuses on the first three stages of their life-cycle: *target selection, exploitation*, and *infection*. Firewalls can block network probes, network intrusion detection systems can detect attempts to exploit vulnerabilities, and host based intrusion detection systems can detect modified, added, and removed files. Despite these efforts, worms can still be designed to avoid detection by these different tools.

Little has been done to detect worms in the fourth stage of their life-cycle, *propagation*. This paper discusses using the standard paradigm of watching for a particular number of events to occur in a given period of time. However, instead of applying this technique to inbound traffic, it is applied to horizontal scans in outbound traffic. This has the distinct advantage of being able to see all the network traffic associated with a particular host. Consequently, worms that were not detected in the first three stages can potentially be detected here.

As the fourth stage occurs after infection, techniques in this area do not explicitly protect the network. Thus, motivations for the use of tools that work in this stage are discussed. These motivations include containing the worm, increased confidence in alerts, and protecting public relations.

More attention should be paid to this phase of the worm's life-cycle, and this paper takes steps in this direction.

References

[1] CERT. Advisory CA-2001-02, Multiple Vulnerabilities in BIND. `http://www.cert.org/advisories/CA-2001-02.html`, January 2001.

[2] CERT. Incident Note IN-2001-03, Exploitation of BIND Vulnerabilities. `http://www.cert.org/incident_notes/IN-2001-03.html`, March 2001.

[3] CERT. Incident Note IN-2001-08, Code Red Worm Exploiting Buffer Overflow in IIS Indexing Service DLL. `http://www.cert.org/incident_notes/IN-2001-08.html`, July 2001.

[4] CERT. Advisory CA-2002-27, Apache/mod_ssl Worm. `http://www.cert.org/advisories/CA-2002-27.html`, September 2002.

[5] M. Fearnow and W. Stearns. SANS Global Incident Analysis Center - Lion Worm Version 0.12. `http://www.sans.org/y2k/lion.htm`, April 2001.

[6] Fyodor. The Art of Port Scanning. `http://www.insecure.org/nmap/nmap_doc.html`, September 1997.

[7] Fyodor. Nmap. `http://www.insecure.org/nmap`, 2001.

[8] R. P. Gorman and E. H. Spafford. Reversing the Network Intrusion Detection Paradigm: The Advantages of Outbound Misuse Detection. *CERIAS Technical Report*, March 2002.

[9] G. H. Kim and E. H. Spafford. The design and implementation of tripwire: A file system integrity checker. In *ACM Conference on Computer and Communications Security*, pages 18–29, 1994.

[10] T. Liston. LaBrea: The Tarpit. `http://www.hackbusters.net/LaBrea/`, 2002.

[11] S. Northcutt, D. McLachlan, and J. Novak. *Network Intrusion Detection: An Analyst's Handbook*. New Riders Publishing, second edition, September 2000.

[12] V. Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks, 31(23-24)*, pages 2435–2463, December 1999.

[13] R. Power. Computer Security Issues & Trends. *Computer Security Institute*, Spring 2001.

[14] T. H. Ptacek and T. N. Newsham. Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection. *Secure Networks, Inc.*, January 1998.

[15] M. Roesch. Snort: The Open Source Network Intrusion Detection System. `http://snort.sourcefire.com/`, 2002.

[16] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, D. Zerkle. Grids–a graph based intrusion detection system for large networks. *The 19th National Information Systems Security Conference*, 1996.

[17] S. Staniford, J. A. Hoagland, and J. M. McAlerney. Practical Automated Detection of Stealthy Portscans. *To appear in the Journal of Computer Security*, 2002. Available at `http://www.silicondefense.com/research/pubs.htm`.

[18] S. Staniford, V. Paxson, and N. Weaver. How to Own the Internet in Your Spare Time. *To Appear in the Proceedings of the 11th USENIX Security Symposium*, August 2002.

[19] G. Stocksdale. NSA Glossary of Terms Used in Security and Intrusion Detection. `http://www.sans.org/newlook/resources/glossary.htm`, April 1998.

Table 1. Worms' Behavior in the Four Stage Life-Cycle

	Target Selection (inbound)		*Exploitation*		*Infection*			*Propagation (outbound)*		
Worm	**Email**	**Network Probe (port #)**	**Targeted Service**	**Vulner. Exploited**	**File-system Mod.**	**Backdoor**	**Other**	**Email**	**Network Probe**	**Other**
Nimda	Yes	Yes (80,600)	IIS, Code Red II and Sadmind Backdoors	CA-2001-06	Yes	No	Yes	Yes	Yes (80,600)	Yes (open network shares, embedded javascript)
Code Red v1	No	Yes (80)	IIS 4.0/5.0, Cisco Series 600 DSL routers	CA-2001-13	Yes	No	No	No	Yes (80)	No
Code Red II	No	Yes (80)	IIS 4.0/5.0, Cisco Series 600 DSL routers	CA-2001-13, IN-2001-09	Yes	Yes	No	No	Yes (80)	No
Adore	No	Yes (21,53,111,515)	BIND, LPRng, rpc.statd, wu-ftpd	CA-2001-02, IN-2001-01	Yes	Yes	Yes	No	Yes (21,53, 111,515)	No
Sadmind	No	Yes (80,111)	IIS, Solstice Sadmind	CA-2001-11, MS00-078	Yes	Yes (port 600)	No	No	Yes (80,111)	No
Lion	No	Yes (53)	BIND	CA-2001-02	Yes	Yes (port 10008)	Yes	No	Yes (53)	No
Ramen	No	Yes (21,111,515)	wu-ftp, rpc.statd, LPRng	IN-2001-01	Yes	Yes	Yes	No	Yes (21,111,515)	No
Cheese	No	Yes (10008)	Lion Backdoor	IN-2001-05	Yes	No	No	No	Yes (10008)	No
Digispid.B	No	Yes (1433)	Microsoft SQL Server	IN-2002-04	Yes	Yes	Yes	No	Yes (1433)	No
Slapper	No	Yes (80,443)	OpenSSL (with Apache running)	CA-2002-27	Yes	Yes (port 2002)	Yes	No	Yes (80,443)	No