UNDERGRADUATE REPORT

Attack Evolution: Identifying Attack Evolution Characteristics to Predict Future Attacks

by MaryTheresa Monahan-Pendergast
Advisor:

UG 2006-6

Attack Evolution:

Identifying Attack Evolution Characteristics To Predict Future Attacks

MaryTheresa Monahan-Pendergast

Dr. Michel Cukier

Dr. Linda C. Schmidt

Dr. Paige Smith

Institute of Systems Research
University of Maryland

ABSTRACT

Several approaches can be considered to predict the evolution of computer security attacks, such as statistical approaches and “Red Teams.” This research proposes a third and completely novel approach for predicting the evolution of an attack threat. Our goal is to move from the destructive nature and malicious intent associated with an attack to the root of what an attack creation is: having successfully solved a complex problem. By approaching attacks from the perspective of the creator, we will chart the way in which attacks are developed over time and attempt to extract evolutionary patterns. These patterns will eventually be used for the prediction of future attacks.

TABLE OF CONTENTS

Attack Evolution:

Identifying Attack Evolution Characteristics to Predict Future Attacks

An attack is a malicious act that attempts to exploit a weakness in a computer system. Such a weakness is known as a security vulnerability. Vulnerabilities are usually classified into host vulnerabilities, network vulnerabilities, and application vulnerabilities. Host vulnerabilities are linked to potential attacks from insiders and lead to potential theft and abuse of privilege (i.e., improper use of authorized operations). Network vulnerabilities are linked to potential attacks from outsiders and lead to potential theft of privilege (i.e., unauthorized increase in privilege). Like host vulnerabilities, application vulnerabilities allow a theft of privilege and an abuse of privilege. An intrusion on a system can be seen as the exploitation of a vulnerability. An intrusion results from an attack that has been, at least partially, successful. An attack is thus an intrusion attempt.

Computers are constantly being attacked. The current state of computer security is mainly reactive. When an attack appears, security companies receive samples of it, proceed to analyze the vulnerabilities being exploited, create patches for these vulnerabilities, and distribute the patches. It is commonly thought that an attack will prosper once a vulnerability is reported and will die off once a patch is distributed. Arbaugh et al. (2000) found that this assumption is false. Actually, patches are often available long before mass exploitations occur. This may be due to the time delay that occurs between the release of a patch and the application of it by a majority of security

administrators. This implies that to effectively defend an organization's network against attacks, administrators should not only check for vulnerabilities, apply patches, and identify and clean infected computers, but should also assess how the attack threat against the organization's network will evolve in order to prevent future attacks.

Several approaches can be considered to predict the evolution of an attack threat. First, data on attacks can be collected and used to predict the attack threat evolution using various statistical approaches (e.g., chronological series). This approach will only be able to predict attack threats based on previous attacks and will not be able to predict the occurrence of any novel attack. Second, red teams (i.e., security experts hired to break into their organization's network) can be used to check for vulnerabilities on the network and how to exploit them. Once a vulnerability is identified, the development team must develop a patch to remove the vulnerability. This approach is very expensive and depends on the quality of the red team. It is also difficult to generalize the results from one organization to another.

This research proposes a third and completely novel approach for predicting the evolution of the attack threat. Currently, there is a strong focus on attacks because of the destructive consequences and the malicious intent. Attacks are considered to be destructive and the attacker is viewed as someone who wants to harm the organization. An attack can be seen as an act of sabotage, but when reviewing sabotage acts, for example against companies, they often happen when someone is very upset or stressed (e.g., someone who has just been fired). A person may try to harm their company by physically destroying some material; however, an employee would rarely sit down and

develop a sophisticated computer attack against the organization's network. Indeed, developing an attack is something very complex, requiring a lot of expertise and an environment favorable to such invention. Our goal is to move from the destructive nature and malicious intent associated with an attack to the root of what attack creation is: having successfully solved a complex problem. Then, attacks can be seen as the successful development of computer programs with, of course, a malicious intent.

In order to study the evolution of attacks, we approached attacks from the perspective of the attack creators. We first studied the hacker society in order to understand what motivates attackers during attack creation. We then reviewed a large set of attacks to understand the types of defining characteristics that change over time. We developed a list of attack characteristics. We selected two families of attacks, Klez and Bagle, and recorded the characteristics of each of the variants. We then attempted to track changes and find evolutionary patterns. This method can be applied to study other attack families as well. Eventually, a new set of recent attacks will be studied and the patterns associated with these attacks will be identified. This should lead to several evolutions of attacks and a method for predicting the types of future attacks that may occur on the Internet.

PROCESS

Hacker R & D

To avoid focusing on the destructive nature of attacks and turn our attention towards attack evolution, we found it helpful to think by way of analogy. Many reports

have been written in this way to create a broader understanding of computer security. Many reports, such as that by Forrest et al.(1997) speak about computer security and attacks in terms of immune systems and biological viruses. These are helpful when thinking about how attacks spread; however, they did not prove to be helpful in guiding our thinking of evolution since they tend to emphasize the destructive effect of attacks. As an article on viral evolution (2004) points out, the danger of thinking in terms of evolution is the implication that attacks change on their own as an effect of their environment. While attacks may evolve, it is important to remember that someone is driving this evolution behind the scenes.

Gaines and Shapiro (1978) describe computer security in terms of criminals and physical security systems. Their report stressed the importance of understanding the attackers' point-of-view. This was helpful to get us thinking about the various approaches one might use to disable or out-smart various types of security mechanisms. The importance of differentiating between premeditated and spur-of-the moment attacks was also stressed. This is a key aspect in determining the type of attacks we will analyze, since only premeditated attacks have the potential for evolution.

We came to think of the hacker community as a team of researchers and developers. Exploratory research is conducted by testing programs, scanning networks, and searching for vulnerabilities. Someone writes an attack intended to exploit a given vulnerability. The attack is the product. Just as developers run tests on their product, the hacker will try the attack program on his own system. Once satisfied, the attack will be

launched. The developer then sits back and waits to see the effect of his product. Alterations will then be made to the product to better reach its goals. Eventually, the developer will attempt to optimize functionality of all aspects of the finished product, thus creating a successful attack.

The Hacker Underground

To understand the way an attack is developed, it is necessary to understand the environment in which this development takes place. We searched for reports that might bring us to an understanding of the hacker sub-culture that exists. One sociological study, conducted by Jordan and Taylor (1998), allowed us to immerse ourselves in this underground society through analyses and interviews with hackers. The primary motivation of hackers does not seem to be the possibility of massive destruction or gain, but rather the pure challenge of breaking into a system. Because of this, information regarding coding, passwords and vulnerabilities is shared openly amongst hackers. An intimate network of information sharing exists through groups, such as *Phrack* and *2600*, magazines, meetings, chat rooms, and online message boards. After accessing the hacker web pages, we were surprised to find that every few years *2600* even holds a large convention, HOPE (Hackers on Planet Earth). By sharing attack knowledge, the group is able to advance collectively at a rapid pace.

Hackers also share information regarding the attacks they achieve in order to gain recognition. Prestige within the hacker community is gained when a hacker creates a noteworthy hack. We contacted Tim Jordan to ask him how a hacker would know that his attack has been successful. Jordan promptly responded that the difficulty is not to

know that one's an attack is successful – this happens once the hacker gains the access or information intended by his attack. Rather, the difficulty is to "prove to others that they were the author of a successful attack so that they can gain peer recognition (personal communication, July 8, 2006)." The successful hackers who receive recognition are those that commit the majority of their time to hacking. Due to the constant flux in information, only those who stay up to date – who we came to refer to as *au courant hackers* – can achieve success. As one hacker stated in an interview,

> If you stop, if you don't do it for one week then things change, the network always changes. It changes very quickly and you have to keep up and you have to learn all the tricks by heart, the default passwords, the bugs you need (Jordan & Paul, 1998).

The motivation of creating novel attacks combined with the constant collaboration within this community strengthens our theory that evolutionary patterns exist.

We can think of hackers of two kinds: experts and amateurs. Experts are the highly skilled people who create innovative attacks. Amateurs will act as copy-cats of the expert attacks. This explains why once a new attack arises, various similar attacks tend to follow. Scripts – programs that allow for attack automation – allow even those with minimal knowledge to create attacks. Typically, once a particular vulnerability is exploited many times, hackers seem to become bored with it and move on. Jordan and Paul also point out that as an attack is duplicated, it loses prestige. We will focus on prominent attacks, in order to see the work of the masters.

Wilson (2001) stresses the distinction between outsider and insider attacks in his report of hacking. Outsiders are responsible for any attack launched on a network from a remote source. An example of an outsider attack is a mass-mailing worm that is launched

onto a system and propagates over various networks. Insiders are responsible for the misuse of legitimate access. An example of an insider is a disgruntled employee who destroys data after being fired or who breaks into an administrator's account to access confidential information. Insiders are responsible for approximately 80% of intrusions (Wilson, 2001); however, since these attackers are typically not involved with the hacker society in which we expect evolution of hacker practices to occur, cases of malicious insider attacks will not apply.

Attack Taxonomy

A great deal of time was spent attaining a proper vocabulary for discussing computer attacks. The difficulty here is that no taxonomy is universally accepted. There are also instances where the line separating different types of malicious code is blurred. We will see an example of this in our Klez analysis, as Klez is a virus-dropping worm. We will assume the following definitions, taken from Hansman and Hunt (2004) throughout the remainder of this report.

- Virus: self-replicating program that propagates by way of infected files
- Worm: self-replicating program, able to propagate without the use of infected files or user interaction
- Trojan: a program made to appear benign that serves some malicious purpose
- Buffer Overflow: a process that gains control or crashes another process by overflowing the portion of memory set aside to store data during transfers
- Denial-of-Service attack: prevents legitimate users from accessing or using a host or network

Classifying & Analyzing Attacks

We initially spent time reviewing and evaluating previous classification methods used in computer security. Hansman and Hunt (2004) set out the requirements for a workable taxonomy. These requirements were helpful as we evaluated other taxonomies. Their own taxonomy used the following characteristics to classify attacks: vector (e.g., virus), target (e.g. Windows XP), vulnerabilities, and exploitations. These categories are seen with minor variations in other reports as well.

Another attack classification we reviewed was defined by Moitra and Konda (2003). In their study they charted the amount of attacks that occurred at sites over a period of time, using the CERT Reports. They then classified the recorded attacks based on: start date, end date, inter-incident time, number of sites, level of incident, method of operation, and type of victim site. From this data they made statistical predictions for the particular sites.

Moore et al. (2001) proposed a method for analyzing particular attacks. Using attack trees, each attack is broken down according to goal, necessary preconditions, steps, and post-conditions. We employed this strategy in order to gain an understanding of the dynamics the following major worms: Melissa, Love Letter, Anna Kournikova, Slammer, Nimda, Klez, Code Red, and Beagle. We used the attack tree technique to look broadly at each of these attacks to see how they functioned. Figure 1 is an example of how we used attack trees on a simple mass e-mailing worm, Melissa.

With a general understanding of worm functioning, we went on to review security reports, provided by McAfee and Symantec, and CERT, as well as case studies on the

worms. In some reports, multiple variants of a worm appeared to be virtually identical. Puzzled by this, and realizing that there must be some reason to assign a new variant name, we sent an e-mail inquiry to the website contacts. Jimmy Kuo, senior McAfee Fellow, explained that an attack will be differentiated as a new variant whenever there is a binary difference, meaning a difference in the actual code. If an attack is compressed, polymorphic, or self-corrupting a new variant name will not be applied (personal communication, July 7, 2006). This information came in very helpful when determining what types of changes were relevant for our purposes.

Code Red is a 2001 worm that attempted and failed a denial-of-service attack on the White House government website. It was programmed so that for one week each month, all infected systems would attempt to access www.whitehouse.gov. The massive amount of traffic was intended to shut down the website. This attack receives a lot of publicity, probably because of its high-profile target; however, the goal was discovered ahead of time and the website was relocated (Moore, 2002). The effective payloads of Code Red were the mass e-mails to slow networks and the defacement of various websites to make the attack known.

The main change that took place between Code Red, Code Red version 2, and Code Red II, seems to be related to its propagation. In order to mass e-mail itself, Code Red would scan IP addresses for vulnerable hosts. In the first version the “random” generator used a static seed, meaning that each time it scanned it would generate the same list of internet protocol (IP) addresses. This stunted its spreading area, as well as, made it easy to track

the course of its propagation. In the second version, a random seed was used to alleviate this problem. Version 2 spread far more efficiently since it attempted more systems and did not waste as much time attempting to infect already infected systems. The IP scan written for Code Red II, the third version, allowed it to spread even further by avoiding most local systems, relative to the infected host, that would probably already be infected. While quite costly to clean up, the Code Red family was not of particular interest to us because of its lack of evolution.

Slammer was an extremely powerful worm launched in 2003. This was the first worm seen to send itself as a UDP packet. Previous worms, such as Code Red, mainly used TCP, transmission control protocol, packets to scan for vulnerable systems and send themselves. TCP is a connection-oriented data transfer system. It is more reliable, but less efficient than UDP because it is latency-limited. Once a TCP packet is sent out there is a waiting period while a response is generated, before more packets can be sent. UDP, user datagram protocol, packets, on the other hand, are connectionless. These packets are bandwidth-limited, meaning that they can send continuous packets as fast as the system allows for, without waiting for any responses (Moore, 2003). Slammer recognized and used this difference in order to maximize its spreading speed.

Nimda was an extremely successful attack. This is because it was a blended attack, spreading by e-mail, shared files, web servers, and IP targeting. The variants of Nimda appear to vary only slightly, probably because it combined various mechanisms from the start. We reviewed SANS White Papers, such as Poore's (2001) report, which analyzed this intricate attack. While Nimda was not a useful attack for this research, it

was helpful to see the way that Nimda writers combined techniques. It seems to us that the authors learned from earlier attacks that capitalized on a particular aspect and combined the mechanisms in one super attack.

A Successful Attack

After an extensive review of these and other attacks, we felt ready to identify the main goals of a hacker when creating a worm. A successful worm attack must be able to propagate quickly, execute successfully, and complete its intended payload. With these goals in mind we were ready to delve into particular attacks to see how these goals are reached over time.

We selected Klez and Bagle to chart and study because each family consisted of many variants. Bakes' (2003) analysis of Klez.H, the most prominent of the Klez family, included a chart that pictured the way each variant propagated and which payload each carried. While these charts did not include the level of detail that we would look for, they did help motivate our original chart. Bakes cited TrendMicro as his data source. This security company was one we had not heard of before, but we found that the reports included more detail than McAfee, Symantec, and CERT. In gathering data on both Klez and Bagle, we used TrendMicro reports, cross-referencing with the other databases. The charts that we have compiled are the first of this kind and should serve as the foundation for identifying attack evolution patterns.

RESULTS

Worm Characteristics

Drawing on the information we collected on various worms, we compiled a list of attach characteristics that we would look at in our charts of Klez and Bagle. We organized this list into the three main categories: Propagation, Execution, and Payload. Propagation refers to mechanisms employed to spread the worm. Execution deals with the way it loads itself and remains on a system. Payload includes what the worm does once it is on a system. Figure 2 is an example of this list. The terms used in the outline are defined here,

- Scan- automated series of targets in order to find which have a certain characteristic, usually to determine which are vulnerable targets
- Encrypted- some change to data, code, or a file that requires it to be processed, or decrypted, before a system can read it
- Compression- minimized in size so that less memory is needed to send or store it
- E-Mail- this is clear, mass e-mailing is the main propagation method of worms; they can be written to steal addresses off the host to mail to, search for vulnerable IP addresses to target, use a spoofed return address to trick recipients, and to use a simple mail transfer protocol (SMTP) or an internet mail access protocol (IMAP)
- File Attachment- e-mails may carry a file attachment containing the worm code; the worm will be executed when the file is downloaded or previewed
- HTML Coded- worms using this mechanism will automatically download the worm from a specified website when the e-mail is opened

- Added to Registry- when a worm writes itself to the system registry key it guarantees that it will be executed each time the system is rebooted
- Written to System Files- worms may write themselves to system files and reside there in memory
- Mutex- this mechanism ensures that only one copy of the worm will run at a time
- Self-Updating- worms are able to access websites to update themselves and stay on a system longer
- Overwrites Files with Zeros- this payload replaces data in files with zeros
- Overwrites Files with Self- this payload replaces data in files with its own code
- Shared File Infection- this payload writes worm code to any shared files on a system so that other users will be infected when they access these
- Disables/Terminates Anti-Virus Programs- this payload stops anti-virus software that is running on the system so that it will not detect the worm's presence
- Deletes Anti-Virus Files- this payload permanently deletes anti-virus programs
- Remote Access- this payload opens up a port or ports, also referred to as a backdoor, on the infected system allowing the attacker access to the system

Worm characteristics are not limited to our list because there are some characteristics that will be specific to certain worms. For example, Klez carries the payload of a "Companion Infection." This characteristic is included in our chart of Klez, but not in the general list because it is specific to Klez. Also, all characteristics in this list will not

be included in all charts because they are not all relevant to every worm. Klez does not involve any mutex mechanisms and so this characteristic is left off of the Klez chart. Characteristics that do not change during an entire family of attacks will also not be shown on our charts. In every version of Klez, for example, the worm wrote itself to the system registry during execution. Since this static characteristic tells us nothing about the evolution of the attack, it is not included in our chart.

Klez

Klez is a virus-carrying worm that has existed in ten different versions. Table 1 is our resulting chart of this family. Klez propagates by mass e-mailing and shared file infections. Authors exploit a vulnerability that causes a file to be executed when previewed in the Microsoft Outlook window. Immediately, the worm loads itself to the system memory and writes itself to the registry. Before e-mailing itself or corrupting system files, Klez lies dormant for a few days. This may be a stealth mechanism, since once the payload is completed, one would be unlikely to link the problem to an e-mail received days before.

Our chart of the Klez family depicts the changes that occurred between variants. Slight changes in propagation techniques are visible. Initially, in versions A and C, the worm was programmed to conduct a scan of all the drives on a host computer. Both of these attempts failed. Klez never attempted a system scan again. Encryption was not consistently used, but was employed in half of the variants. All but two accomplished the mass e-mailing procedure. Klez.J attempted and failed, but was also written to infect shared files, allowing it to propagate in this way. Version B does not attempt to mass

e-mail or to infect shared files, leaving it with no hope of propagation. Actually, Klez.B appears to serve the sole purpose of gaining remote access – the only in the family with this payload. Unlike its siblings, it does not mass e-mail, infect files, or affect anti-virus software. We speculate that the first three versions were attempts to gain information about systems before launching full blown attacks.

Klez harvests e-mail addresses to send itself to from the Windows Address Book (WAB) on the host computer. In three variants it also retrieves them from 17 other files, in an attempt to spread to even more systems with each infection. The address appearing in the "From:" field of e-mail also changed between versions. Initially, a random address found in the worm body was used. Later, to avoid suspicion, Klez would spoof the "From:" address with the address of an uninfected user found on the host's computer. For example, if Sue was infected by Klez, and Bob and Harry are both in her address book, an e-mail sent out from Sue's computer will go to Harry and others, but it will appear to come from Bob. While Bob may be harassed about sending out a worm that didn't actually come from him, Sue may still be unaware that she is infected. Early on, propagation by e-mail employed one of the simple mail transfer protocol (SMTP) engines embedded in the e-mail, namely yahoo, hotmail, or sina. Later it narrowed this to send with the same engine of the spoofed return address. By the later versions, Klez was written to retrieve and use the SMTP engine that was used by the host computer.

The payload of Klez is the dropping of the ElKern virus. ElKern is a polymorphic virus, meaning that it evades detection by automatically modifying its code every time it replicates. When it is dropped ElKern infects files, copies itself into the Windows

System folder, and encrypts itself. It also writes itself to the registry so that it will run, infecting random files, every time the system is rebooted. Early Klez variants carried the first version of ElKern, which overwrote the information contained in files with zeros while maintaining the original file sizes to avoid detection. The second version, ElKern.B, was virtually the same in effect, but included an algorithm to avoid infecting self-infecting .rar and .zip files. Self-extracting files are able to extract themselves each time they run. This would seem to make the virus visible anytime an infected file was run. Later, ElKern.C is dropped, but this third version contains no destructive payload. It does not write itself to the registry and it does not infect files. The final Klez variant returns to dropping the previous (destructive) virus, ElKern.B.

Most versions of Klez also perform the task of infecting shared files on a system. Whenever another user accesses these shared files, they will be infected, providing a second propagation mechanism. Klez enables itself to propagate undetected for spans of time by attacking anti-virus software on infected hosts. The first Klez disables anti-virus programs. Version C is written to disable them each time the system reboots. The next variant disables anti-virus programs and also removes them from the registry so that they will fail to run upon later reboots. Beyond that, with the exception of Klez.J, anti-virus programs are disabled and entirely deleted from the system.

The last three versions of Klez contain a payload referred to as a companion infection. This type of infection first compresses a targeted executable file, replacing the extension .EXE with a random extension. It then sets the file properties to Read-only,

Hidden, System, and Archive. These settings keep the clean file from being easily visible to the user. The worm then copies itself into the original filename so that the host will run it.

Klez.H was the most successful variant, infecting over 3 million systems since its release on April 17, 2002. Its authors optimized their product in this version, spreading to an enormous area. Propagation advances included spoofing the "From:" field and using the SMTP engine of the host, in order to encourage more recipients to open the message. It also infected shared files to provide a second propagation method. This version not only disabled, but also totally deleted anti-virus programs on the infected system to avoid being detected. While it dropped the less effective ElKern version C, it employed the unique Klez companion infection in order to infect and remain undetected on various areas of the system memory.

Overall, it appears that Bagle authors initially optimized the way the e-mail was sent, focusing on addresses, spoofing, and engines. Next, they attempted to increase execution success by using the improved version of ElKern. The third focal point was detection avoidance, which they dealt with first by optimizing the anti-virus attack and then by developing the companion infection payload.

Bagle

Bagle, referred to by some as Beagle, is a mass e-mailing worm seen in sixty-two versions. Our chart of the Bagle family has been broken down into Table 2, variants A through Z, Table 3, variants AB through AZ, and Table 4, variants BA through BY for viewing purposes. One explanation for the large number of variants is that each Bagle

is written to infect computers within a relatively short time frame. For example, the first variant was reported on January 18, 2004, and is written to be active only through January 28, 2004. If it reaches a computer on or after the latter date, it terminates itself. This mechanism also allows Bagle better chances of avoiding detection. Using our chart we are able to recognize some trends in the formulation of Bagle. We can also recognize various characteristics that authors played with a few times, but ultimately abandoned.

During propagation, Bagle seems to favor compression over encryption as a stealth mechanism. Once executed, Bagle scans for vulnerable IP addresses to target. In about a third of the versions, an external domain name system (DNS) is accessed at a particular IP address in order to resolve vulnerable addresses. Over time this system is located at three different locations.

All Bagle variants attempted mass e-mailing. Address harvesting changes significantly over the course of the Bagle family as authors try different approaches. Initially, Bagle is written to access only four types of files in search of addresses. At times it is written to search as many as thirty-one different types of files. At one point, Bagle tries to access only two file types, with version AF. Later, in a very unsuccessful attempt, Bagle.BN accesses the file P:\Emails only. Towards the end of the family, variants tend to access only the Windows Address Book (WAB). Four variants do not harvest any addresses from the host, but instead access a list of addresses on a particular website. A few versions combine the WAB addresses with this internet list. All versions are written to avoid sending to any addresses that contained certain strings, such as

"@symantec" and "admin," not wanting to send to the wrong people – those trained to detect a worm.

Early on Bagle e-mails itself with a particular spoofing technique. When it harvests a list of addresses, say on Sue's computer, the first e-mail might appear to be sent from Bob to Bob, the second from Bob to Harry, the third from Harry to Martha, and so on. E-mails never appeared to come from the actual infected user. This spoofing technique also makes it extremely difficult to track propagation.

Typically Bagle e-mails contain a file attachment carrying the worm code. In many versions authors use password-protected files. The password is contained within the body of the e-mail. This is probably intended to lure the recipients to open the file. A password-protected file may seem to be more trustworthy or simply enticing to users. At one point, in variant X, authors attach the infected file along with a clean file, again, probably as bait for recipients. In version Q, Bagle began exploiting a PopUp Window vulnerability that allowed the worm to be downloaded from a website once the HTML-coded e-mail was viewed. HTML-coding is first employed along with file attachments, then alone, and eventually authors return to using file attachments alone.

Mutexes are programs that allow only one program of similar type to run at a time. Early on, a mutex is included in the code of many Bagle versions in order to block two copies of itself from running at a time. This is most likely a way to avoid detection, considering that the memory used to run multiple copies of worm code would slow down a system tremendously. Many of the Bagle variants are also able to update themselves by

accessing a website and running the UPD command. They would also access a website and send information to the hacker to notify him of systems he has reached.

A main payload of Bagle is the achievement of remote access on a system. Over time the targeted TCP port changes. Versions AF, AH, and AJ also open remote access on a UDP port. Beginning with Bagle.F, over half of the variants infect all files whose names include the string "shar," allowing propagation to occur over network shares. All versions of Bagle infect system files, in order to remain in memory. Versions N through T also infect executable files on the system. Version T compresses the executable files once it infects them. This payload may be destructive, but it is not desirable enough to be repeated on many later versions. Bagle.BB modifies HOST files so that a user is unable to access anti-virus and security updating websites. It is also written to monitor all internet browsing done on the infected system. Only six versions of Bagle do not attack anti-virus software. All others disable the running programs and version AE even deletes the programs from memory. More than half of the variants also attempt to access a large number of websites. This payload may cause internet traffic and slow down networks.

Over a third of the Bagle variants combine programs by carrying a Trojan or virus. Five versions carried with them a polymorphic virus and twenty carried a trojan. These versions affect systems as the others, but they work in conjunction with the additional programs with Bagle being mainly responsible for the propagation.

By version N, reported on March 3, 2004, less than three months after the first report of Bagle, a new trend in the family appears. Bagle code began to attack the worm NetSky. This worm, which is very similar to Bagle, was first reported about a month

after the first Bagle. The two have since engaged in an "internet skirmish", according to reports at Sophos (2004). Most Bagle versions, following N, deletes NetSky code from infected systems. Bagle.AC deleted not just NetSky, but also the mutex that NetSky installs. AD installs a mutex, as earlier Bagle versions had. This mutex, however, was intended to block NetSky, not Bagle, code from running.

It seems that the focus of Bagle authors changed over the course of this worm. Initially, the e-mail propagation underwent changes to the sources for addresses harvesting and spoofing techniques. Eventually, authors also had a focused list of recipients that they consistently used. Relatively early on authors perfected the shared file infection and spread technique. Then, changes were made to the mode of execution. Authors tried password protection on file attachments for quite awhile and eventually attempted HTML-coded e-mails as well. Stealth approaches, including compression, anti-virus attacks, and mutexes, were refined over time. Once the worm seemed to be functioning well, attention was placed on the emerging NetSky battle.

Towards the end of the Bagle family charted here, variants seemed to level out, probably to what the authors are finding to be most effective for their current goals. The seemingly ideal Bagle is an e-mail carrying a compressed file to addresses stored on a designated website and in the user's WAB. Its main goal appears to be to stop NetSky from running on the infected system, disable anti-virus software, and to set up remote access.

The Driving Force

Within the categories of Propagation, Execution, and Payload, we can see what

characteristics seem to drive changes. Regardless of its intended payload, the main goal of a worm is to spread. This goal affects all three categories. Within propagation, changes are made to find as many vulnerable hosts as possible. Changes are made in selecting harvesting sites as well as in what techniques are used to hide the identity of the worm. Compression and spoofing are important factors. Within execution, changes are made to the way code is written to a system. In order to continue spreading, the worm must achieve infection. Again, authors play with the infection methods to find what works best to ensure the worm is executed. Payloads vary, from destructing files to opening systems, but one thing that appears to be crucial is anti-virus attacks. In order to maximize its spreading ability, a worm must be able to remain resident on a system without being detected long enough find future hosts and begin spreading to them.

It all comes down to propagation. Worms must be capable of spreading quickly, but also to a large area. For instance, if a worm spread very quickly, but only hit a few networks, it would be stopped once administrators in these networks patched the vulnerability. By spreading quickly and to distant networks, a worm can evade patches longer. If a worm is written to spread quickly, but is easily recognized as a worm, either by anti-virus detection or by individual users, it will also not spread and so various stealth mechanisms are needed. It is clear that authors try out different combinations, learn from mistakes, and replicate successes in order to achieve powerful propagation. Once propagation is optimized, hackers may add their desired payloads, ultimately developing a successful product.

LIMITATIONS of this RESEARCH

In researching these attacks, we found that there are discrepancies between company reports. Not all databases will agree upon variant names. For example, Bagle.AC reported by Trend Micro on August 9, 2004, is recorded on the Symantec database as Bagle.AO, and on McAfee as Beagle.AQ. As stated, we used TrendMicro databases as our main source of information; therefore, our names and dates are aligned with these reports. The inconsistency among sources poses a challenge to future research in this area. There are also noticeable holes in our charts. The explanation, for example, for not including Bagle.AP, is simply that no report exists.

In order to find evolutionary patterns, more families than will need to be researched. Any trends or patterns stated here are speculations at this point. More data will need to be gathered in order to draw valid conclusions. It should also be noted that within the Bagle family, variants do not necessarily fall in chronological order. For example, Bagle.AO was actually reported six months after the BY version. Variant U does not fall in line with the progression of the other variants, as it appears to be an original Bagle prototype. The reason for these inconsistencies is also not clear; however the time delay in reporting should be noted. The dates listed on our charts are the dates that TrendMicro received samples of the attacks and began analyzing them. The dates that the attacks were originally released by the hackers are unknown.

It should be noted that further Bagle variants exist beyond the versions we charted. After completing the Bagle chart of variants A through BY, we came across evidence that Bagle may actually be a single piece in a larger network of attacks. Molenkamp and

O'Dea (2005) revealed that the Bagle we studied served the purpose of mass mailing in a larger scheme, pictured in Figure 3. Another piece serves to download from the internet and another to gather information. This may explain why certain features were abandoned over time. For example, we found that the payload of remote access dropped off in time; however, this may be because Mitglieder was imposed in the Bagle network to serve this distinct purpose. Hackers are assumed to be making profit off this scheme by using many weakly protected computers as launch sites for destructive attacks on companies.

FUTURE RESEARCH

To continue with this project, we would recommend charting the NetSky family and conducting a comparative study between this and the Bagle family. Because the two are caught in a competition, it will be useful to examine which approaches have been successful in putting one ahead of the other in the worm race. Combining will allow for insight into how the hackers learn, not only from their own mistakes and successes, but also from each other's. As further case studies are done on this competition, it will be interesting to see what the motivation is. Are the attackers caught in an originality draught, forced to copy each other? Are they caught up in the rush of competition? Or, is this cyber war intended to distract analysts from the bigger picture – the suspected underground of thousands of infected machines? Understanding this might lend further insight into how these attackers learn and play off of each other.

While we have not defined specific evolutionary patterns, our research has shown that these patterns may, and probably do, exist. Our charts are the first step in approaching families of attacks in this way. Due to time restraints on this research, further analysis of these charts has not yet been completed. Charts of other families should be created using our worm characteristics. Diagramming and probabilistic analyses can then be applied to all of the charts in order to extract evolutionary patterns and predictions about the future of this evolution.

REFERENCES

2600. Retrieved on June 13, 2006, from http://www.2600.org.

Arbaugh, W. A., Fithen, W. L., & McHugh, J., (2000). Windows of Vulnerability: A Case Study Analysis. *IEEE*, 2000. Retrieved June 13, 2006, from IEEE Distributed Systems Online.

Bagle and Netsky Battle for Supremecy in Internet Skirmish, Sophos Advices (2004, March 3). Retrieved on July 24, 2006, from http://www.sophos.com/pressoffice/news/articles/2004/03/va_wormwar.html.

Bakes, M., (2003). Klez.H: From Propagation to Prevention. *SANS White Papers.* Retrieved on July 20, 2006, from the Information Security Reading Room.

Berghel, H., (2001). The Code Red Worm: Malicious Software Knows No Bounds. *Communications of the ACM,* 44.12, 15-19. Retrieved June 22, 2006, from the ACM Digital Library.

Forrest, S., Hofmeyr, S.A., & Somayaji, A., (1997). Computer Immunology. *Communications of the ACM, 40.10, 88-88-96.* Retrieved on June 8, 2006, from the ACM Digital Library.

Gaines, R. S., & Shapiro, N. Z., (1978). *ACM SIGOPS Operating Systems Review.* Retrieved June 7, 2006, from ACM Digital Library.

Hansman, S., & Hunt, R., (2004). A Taxonomy of Network and Computer Attacks. *Computers & Security*, 24, 31-43. Retrieved on June 8, 2006, from ElSevier.

Hill, M., (2002). Blended Threats: The New Security Vulnerability. *SANS Secuirty Essentials Practical Assignment.* Retrieved June 20, 2006, from GIAC Practical Repository.

Jordan, T., & Taylor, P., (1998). A Sociology of Hackers. *The Sociological Review.* Retrieved June 8, 2006, from MetaLib Research Port.

Moitra, S. D., & Konda, S. L., (2003). An Empirical Investigation of Network Attacks on Computer Systems. *Computers & Security*, 23, 43-51. Retrieved on June 8, 2006, from ElSevier.

Molenkamp, S., & O'Dea, Hamish (2005, October). Solving the Bagle Jigsaw. Retrieved on July 19, 2006, from the Virus Bulletin Conference.

Moore, A. P., Ellison, R. J., & Linger, R. C., (2001) Attack Modeling for Information Security and Survivability. Retrieved June 7, 2006, from Carnegie Mellon University.

Moore, D., Paxson, V., Savage, S., Shannon, C., Staniford, S., & Weaver, N., (2003). Inside the Slammer Worm. *IEEE Security & Privacy*. Retrieved June 21, 2006, from IEEE Distributed Systems Online.

Moore, D., Shannon, C., & Claffy, K., (2002). Code-Red: A Case Study on the Spread and Victims of an Internet Worm. Retrieved June 24, 2006, from CAIDA, San Diego Supercomputer Center, University of California.

Phrack. Retrieved on June 13, 2006, from http://www.Phrack.org.

Poore, K., (2001). Nimda Worm – Why is it Different? *SANS White Papers*. Retrieved June 21, 2006, from the Information Security Reading Room.

Security of the Internet (1997). Retrieved June 6, 2006, from http://www.cert.org/encyc_article/tocencyc.html.

Trend Micro Security Information. Retrieved in July-August 2006, from http://www.trendmicro.com/vinfo/default.asp?sect=SA.

Viral Evolution (2004, December). Retrieved on July 18, 2006, from http://www.infectionvectors.com/vectors/virolution.htm.

Wilson, Zachary (2001). Hacking: The Basics. *SANS White Papers.* Retrieved July 6, 2006, from the Information Security Reading Room.

Table 1. Klez Family

		Propagation						Execution	Payload					
Variant	**Date of Discovery**	**Scan**	**Encrypted**	**Mass E-Mail**	**SMTP engine**	**"From:"**	**Addresses**	**Drops when Previewed**	**Shared File Infection**	**Anti-Virus Disabled**	**Anti-Virus Deleted**	**Overwrites Files with Zeros**	**Companion Infection****	**Remote Access**
Klez.A	10/25/2001	Yes (a-z) failed	Yes	Yes	yahoo, hotmail, sina	worm body	WAB	W.32.ElKern.A	Yes	Yes	No	Yes (fixed & remote) failed	No	No
Klez.B	10/30/2001	No	No	No					No	No	No	No	No	Yes
Klez.C	11/15/2001	Yes (a-z) failed	Yes	Yes	yahoo, hotmail, sina	worm body	WAB	W.32.ElKern.A	Yes	Yes (at every reboot)	No	Yes (fixed & remote) failed	No	No
Klez.D	11/8/2001	No	No	Yes	yahoo, hotmail, sina	worm body	WAB	W.32.ElKern.A	No	Yes (& removed from registry)	No	Yes (fixed & shared) failed	No	No
Klez.E	1/17/2002	No	Yes	Yes	domain of from: field	worm body	WAB & other files*	W.32.ElKern.B	Yes	Yes	Yes	Yes	No	No
Klez.F	1/22/2002	No	Yes	Yes	domain of from: field	worm body (vary from E)	WAB & other files	W.32.ElKern.B	Yes	Yes	Yes	Yes	No	No
Klez.G	4/17/2002	No	No	Yes	smtp in registry	uninfected in host's files	WAB	W.32.ElKern.B	Yes	Yes	Yes	No	No	No
Klez.H	4/17/2002	No	Yes	Yes	smtp in registry	uninfected in host's files	WAB & other files	W.32.ElKern.C	Yes	Yes	Yes	No	Yes	No
Klez.I	4/20/2002	No	No	Yes	smtp in registry	uninfected in host's files	WAB	W.32.ElKern.C	Yes	Yes	Yes	No	Yes	No
Klez.J	9/26/2002	No	No	No	failed	uninfected in host's files	WAB	W.32.ElKern.B	Yes	No	No	No	Yes	No

*also extracts addresses from the following files: MP8, EXE, SCR, PIF, BAT, TXT, HTM, HTML, DOC, XLS, CPP, C, PAS, MPQ, MPEG, BAK, MP3

**This type of infection first compresses the target .EXE file, replacing .EXE with a random extension. It sets the file properties to Read-only, Hidden, System, and Archive for stealth purposes. The worm then copies itself into the original filename.

Table 2. Bagle Family (A-Z)

		Propagation						Execution							Payload				
Variant	Date of Discovery	Compressed	Encrypted	Polymorphic Virus	External DNS Server Access	Downloads List of Banned IP's	Harvests Addresses	File Attachment	Password Protected	HTML code	Mutex	Deletes NETSKY	Trojan	Self-Updating	Shared File Infection	Fixed File Infection	Anti-Virus Disabled	Remote Access	Accesses Websites
Bagle.A	1/18/2004	No	No	No	No	No	Yes (I)	Yes	No	No	No	No	No	No	No	No	No	Yes (port 6777)	attempted
Bagle.B	2/17/2004	No	No	No	No	No	Yes (I)	Yes	No	No	No	No	No	Yes	No	No	No	Yes (port 8866)	attempted
Bagle.C	2/27/2004	Yes	No	No	Yes (151.201.0.39)	No	Yes (II)	Yes	No	No	Yes (imain_mutex)	No	No	Yes	No	No	Yes	Yes (2745)	Yes
Bagle.D	2/28/2004	Yes	No	No	Yes (151.201.0.39)	No	Yes (II)	Yes	No	No	Yes (iain_m2)	No	No	Yes	No	No	Yes	Yes (2745)	Yes
Bagle.E	2/28/2004	Yes	No	No	Yes (151.201.0.39)	No	Yes (II)	Yes	No	No	Yes (imain_mutex)	No	No	Yes	No	No	Yes	Yes (2745)	Yes
Bagle.F	2/29/2004	Yes	No	No	Yes (151.201.0.39)	No	Yes (III)	Yes	Yes	No	Yes (imain_mutex)	No	No	Yes	Yes (shar)	No	Yes	Yes (2745)	Yes
Bagle.G	2/29/2004	Yes	No	No	Yes (151.201.0.39)	No	Yes (III)	Yes	Yes	No	Yes (imain_mutex)	No	No	Yes	Yes	No	Yes	Yes (2745)	Yes
Bagle.H	3/1/2004	No	No	No	Yes (151.201.0.39)	No	Yes (III)	Yes	Yes	No	Yes (imain_mutex)	No	No	No	Yes	No	Yes	Yes (2745)	No
Bagle.I	3/2/2004	No	No	No	Yes (151.201.0.39)	No	Yes (III)	Yes	Yes	No	Yes (imain_mutex)	No	No	No	Yes	No	Yes	Yes (2745)	No
Bagle.J	3/2/2004	Yes	No	No	Yes (151.201.0.39)	No	Yes (III)	Yes	Yes	No	Yes (imain_mutex)	No	No	Yes	Yes	No	Yes	Yes (2745)	Yes
Bagle.K	3/3/2004	Yes	No	No	Yes (151.201.0.39)	No	Yes (III)	Yes	Yes	No	No	No	No	Yes	Yes	No	Yes	Yes (2745)	Yes
Bagle.L	3/9/2004	Yes	No	No	No	Yes	Yes (III)	Yes		No	No	No	Yes	No	No	No	Yes	Yes (11117, 35555)	Yes
Bagle.M	3/13/2004	Yes	No	No	Yes (151.201.0.39)	Yes	Yes (III)	Yes		No	Yes (imain_mutex)	No	Yes	No	No	No	Yes	Yes (random)	Yes
Bagle.N	3/13/2004	Yes	No	Yes	Yes (217.5.97.137)	Yes	Yes (IV)	Yes	Yes	No	No	Yes	No	No	Yes	Yes (all .EXE)	Yes	Yes (2556)	No
Bagle.O	3/13/2004	Yes	No	Yes	Yes (217.5.97.137)	No	Yes (IV)	Yes	Yes	No	No	Yes	No	Yes	Yes	Yes (all .EXE)	Yes	Yes (2556)	No
Bagle.P	3/14/2004	Yes	No	Yes (&pads w.garbage)	Yes (217.5.97.137)	No	Yes (IV)	Yes	Yes	No	No	Yes	No	Yes	Yes	Yes (all .EXE)	Yes	Yes (2556)	No
Bagle.Q	3/17/2004	Yes	Yes	Yes	Yes (217.5.97.137)	No	Yes (IV)	attempted	Yes	Yes	No	Yes	No	No	Yes	Yes (all .EXE)	Yes	Yes (2556)	No
Bagle.R	3/18/2004	Yes	Yes	No	No	No	Yes (IV)	attempted	Yes	Yes	No	Yes	No	No	Yes	Yes (all .EXE)	Yes	Yes (2556 & random)	No
Bagle.S	3/18/2004	Yes	No	Yes	No	No	Yes (IV)	attempted	Yes	Yes	No	Yes	No	Yes	Yes	Yes (all .EXE)	Yes	Yes (2556 & random)	No
Bagle.T	3/18/2004	Yes	Yes	No	No	No	Yes (IV)	Yes	Yes	Yes	No	No	No	No	No	Yes (&compresses)	Yes	Yes (2556)	No
Bagle.U	3/26/2004	Yes	No	No	No	No	Yes (IV)	Yes	No	No	Yes	No	No	Yes	No	No	No	Yes (4751)	No
Bagle.V	3/29/2004	Yes	No	No	No	No	Yes (IV)	Yes	No	No	Yes	No	No	Yes	No	No	No	Yes (4751)	No
Bagle.W	4/26/2004	Yes	No	No	No	Yes	Yes (IV)	Yes	Yes	No	No	No	Yes	No	Yes	No	Yes	Yes (17771, 35555)	Yes
Bagle.X	4/26/2004	No	No	No	No	No	Yes (IV)	Yes (two, one clean)	Yes	No	No	Yes	No	No	Yes	No	Yes	Yes (2535)	No
Bagle.Y	4/25/2006	Yes	No	No	Yes (217.5.97.137)	Yes	Yes (IV)	Yes		No	No	No	Yes	No	No	No	Yes	Yes (18881)	Yes
Bagle.Z	4/29/2004	No	Yes	No	No	No	Yes (IV)	Yes	Yes	No	No	Yes	No	No	Yes	No	Yes	Yes (2535)	No

I: WAB, TXT, HTM, HTML
II: WAB, TXT, HTM, HTML, ADB, ASP, CFG, DBX, EML, MDX, MMF, NCH, ODS, PHP, PL, SHT
III: WAB, TXT, HTM, HTML, ADB, ASP, CFG, DBX, EML, MDX, MMF, NCH, ODS, PHP, PL, SHT, XML
IV: WAB, TXT, HTM, HTML, ADB, ASP, CFG, DBX, EML, MDX, MMF, NCH, ODS, PHP, PL, SHT, XML, CGI, DHTM, JSP, MBX, MHT, MSG, OFT, SHTM, STM, TBB, UIN, WSH, XLS, SML

Table 3. Bagle Family (AB-AZ)

		Propagation						Execution							Payload				
Variant	Date of Discovery	Compressed	Encrypted	Polymorphic Virus	External DNS Server Access	Downloads List of Banned IP's	Harvests Addresses	File Attachment	Password Protected	HTML code	Mutex	Deletes NETSKY	Trojan	Self-Updating	Shared File Infection	Fixed File Infection	Anti-Virus Disabled	Remote Access	Accesses Websites
Bagle.AB	5/9/2004	No	No	No	No	No	Yes (IV)	Yes	Yes	No	No	Yes	No	No	Yes	No	No	Yes (2535)	No
Bagle.AC	8/9/2004	Yes	No	No	No	No	Yes (IV)	No	n/a	Yes	No	Yes (&its mutex)	Yes	No	Yes	No	Yes	No	No
Bagle.AD	7/4/2004	Yes	No	No	No	No	Yes (IV)	Yes	Yes	No	Yes (to avoid NetSky)	Yes	No	Yes	Yes	No	No	Yse (1234)	No
Bagle.AE	7/8/2004	No	No	No	No	No	Yes (IV)	Yes	Yes	No	Yes	Yes	No	No	Yes	No	deletes from registry	Yes (1234)	No
Bagle.AF	7/15/2004	Yes	Yes	No	No	No	Yes (DHTM, SHTM)	Yes	Yes	No	Yes	Yes	No	Yes	Yes	No	Yes	Yes (1080 & rand UDP)	No
Bagle.AG	7/17/2004	Yes	No	No	No	No	Yes (IV)	Yes	Yes	No	Yes	Yes	No	No	Yes	No	Yes	No	No
Bagle.AH	7/19/2004	No	No	No	No	No	Yes (IV)	Yes	Yes	No	Yes	Yes	No	No	Yes	No	Yes	Yes (1080 & rand UDP)	No
Bagle.AI	8/31/2004	Yes	No	No	No	No	Yes (IV)	No	n/a	Yes	Yes	Yes	Yes	No	Yes	No	Yes	No	No
Bagle.AJ	8/16/2004	No	Yes	No	No	No	Yes (IV)	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes (random UDP&TCP)	No
Bagle.AK	8/31/2004	Yes	No	No	No	No	Yes (IV)	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	No	Yes	No	No
Bagle.AL	8/31/2004	No	No	No	No	No	Yes (IV)	No	n/a	Yes	Yes	Yes	Yes	Yes	Yes (shar)	No	No	No	No
Bagle.AM	9/28/2004	Yes	No	No	No	No	Yes (IV)	Yes	No	No	Yes	Yes	No	No	Yes (shared)	No	Yes	No	Yes
Bagle.AN	10/29/2004	No	No	No	No	No	Yes (IV)	Yes	No	No	Yes	Yes	No	No	Yes (shar)	No	Yes	Yes (81)	Yes
Bagle.AO	6/29/2006	Yes	No	No	No	No	No (from URL)	Yes	No	No	Yes	No	No	No	No	No	No	No	Yes
Bagle.AT	10/29/2004	Yes	Yes	No	No	No	Yes (IV)	Yes	No	No	Yes	No	No	Yes	Yes	No	Yes	Yes (81)	Yes
Bagle.AU	10/29/2004	No	No	No	No	No	Yes (IV)	Yes	No	No	Yes	Yes	No	No	Yes	No	Yes	Yes (81)	Yes
Bagle.AX	11/17/2004	Yes	No	No	No	No	Yes (IV)	Yes	Yes	No	Yes	No	No	No	Yes	No	Yes	Yes (2002)	Yes
Bagle.AY	1/26/2005	No	No	No	Yes (17.5.97.137)	No	Yes (IV)	Yes	No	No	Yes	Yes	No		Yes	No	Yes	No	Yes
Bagle.AZ	1/25/2005	Yes	No	No	Yes (17.5.97.137)	No	Yes (IV)	Yes	No	No	Yes	Yes	No	No	Yes	No	Yes	Yes (81)	Yes

I: WAB, TXT, HTM, HTML
II: WAB, TXT, HTM, HTML, ADB, ASP, CFG, DBX, EML, MDX, MMF, NCH, ODS, PHP, PL, SHT
III: WAB, TXT, HTM, HTML, ADB, ASP, CFG, DBX, EML, MDX, MMF, NCH, ODS, PHP, PL, SHT, XML
IV: WAB, TXT, HTM, HTML, ADB, ASP, CFG, DBX, EML, MDX, MMF, NCH, ODS, PHP, PL, SHT, XML, CGI, DHTM, JSP, MBX, MHT, MSG, OFT, SHTM, STM, TBB, UIN, WSH, XLS, SML

Table 4. Bagle Family (BA-BY)

		Propagation						Execution							Payload				
Variant	Date of Discovery	Compressed	Encrypted	Polymorphic Virus	External DNS Server Access	Downloads List of Banned IP's	Harvests Addresses	File Attachment	Password Protected	HTML code	Mutex	Deletes NETSKY	Trojan	Self-Updating	Shared File Infection	Fixed File Infection	Anti-Virus Disabled	Remote Access	Accesses Websites
Bagle.BA	2/8/2005	No	No	No	No	No	Yes (IV)	Yes	No	No	Yes	Yes	No	Yes	Yes	No	Yes	Yes (81)	Yes
Bagle.BB	8/8/2005	No	No	No	No	No	Yes (WAB)	No	n/a	Yes	Yes	Yes	Yes	Yes	Yes	Yes (HOST files)	No	No	Yes
Bagle.BD	7/25/2005	No	No	No	Yes (217.5.97.137)	No	Yes (WAB & URL)	Yes	Yes	No	Yes	Yes	Yes	No	No	No	No	Yes (80)	Yes
Bagle.BE	2/28/2005	Yes	No	No	Yes (217.5.97.137)		No (from URL)	Yes	No	No	Yes	Yes	Yes	No	No	No	Yes	Yes (80)	Yes
Bagle.BG	3/4/2005	Yes	No	No	Yes (217.5.97.137)	No	Yes (WAB & URL)	Yes	No	No	Yes	No	Yes	No	No	No	Yes	Yes (80)	Yes
Bagle.BH	4/18/2005	Yes	No	No	No	No	No (from URL)	Yes	Yes	No	Yes	No	Yes	No	No	No	Yes	No	Yes
Bagle.BI	4/18/2005	Yes	No	No	No	No	Yes (WAB)	Yes	Yes	No	Yes	No	Yes	No	No	No	Yes	No	Yes
Bagle.BM	8/4/2005	Yes	No	No	No	No	Yes (IV)	Yes	Yes	No	No	No	No	Yes	Yes	No	Yes	Yes (9030)	Yes
Bagle.BN	8/4/2005	Yes	No	No	No	No	Yes (P:\E-mails)	No	n/a	Yes	Yes	Yes	Yes	No	Yes	No	Yes	No	Yes
Bagle.BQ	11/1/2005	Yes	No	No	No	No	Yes (WAB)	Yes	Yes	No	No	Yes	Yes	No	No	No	Yes	No	No
Bagle.BR	10/23/2005	Yes	No	No	No	No	Yes (WAB)	Yes	Yes	No	Yes	No	Yes	No	No	No	Yes	No	Yes
Bagle.BS	11/2/2005	Yes	No	No	No	No	Yes (WAB)	Yes	Yes	No	Yes	No	Yes	No	No	No	Yes	No	Yes
Bagle.BT	10/6/2005	Yes	No	No	No	No	Yes (WAB)	Yes	No	No	Yes	No	No	Yes	Yes	No	Yes	No	Yes
Bagle.BX	11/23/2005	Yes	No	No	No	No	Yes (WAB & URL)	Yes	Yes	No	Yes	No	No	No	No	No	Yes	No	Yes
Bagle.BY	12/25/2005	Yes	No	No	No	No	No (from URL)	Yes	No	No	Yes	No	Yes	Yes	No	No	Yes	Yes (80)	Yes

I: WAB, TXT, HTM, HTML
II: WAB, TXT, HTM, HTML, ADB, ASP, CFG, DBX, EML, MDX, MMF, NCH, ODS, PHP, PL, SHT
III: WAB, TXT, HTM, HTML, ADB, ASP, CFG, DBX, EML, MDX, MMF, NCH, ODS, PHP, PL, SHT, XML
IV: WAB, TXT, HTM, HTML, ADB, ASP, CFG, DBX, EML, MDX, MMF, NCH, ODS, PHP, PL, SHT, XML, CGI, DHTM, JSP, MBX, MHT, MSG, OFT, SHTM, STM, TBB, UIN, WSH, XLS, SML

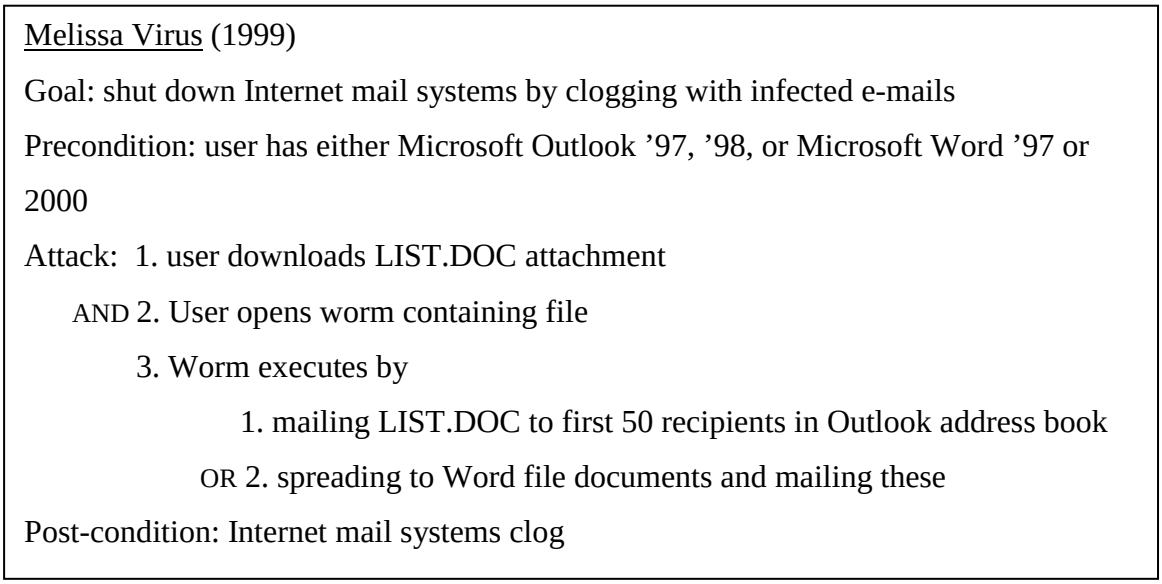

Figure 1. Melissa Virus Attack Tree

- **Propagation**
  - Scan
    - drive(s)
    - UDP/TCP
  - Encrypted
  - Compressed
  - E-Mail
    - Harvests Addresses
    - Target IP's
    - "From:"
    - SMTP/IMAP

- **Execution**
  - File Attachment
    - attachment previewed
    - attachment opened
  - HTML Coded
  - Added to Registry
  - Written to System Files
  - Mutex
  - Self-Updating

- **Payload**
  - Overwrites Files with Zeros
  - Overwrites Files with Self
  - Shared File Infection
  - Disables/Terminates Anti-Virus Programs
  - Deletes Anti-Virus Files
  - Remote Access

Figure 2. Worm Characteristics

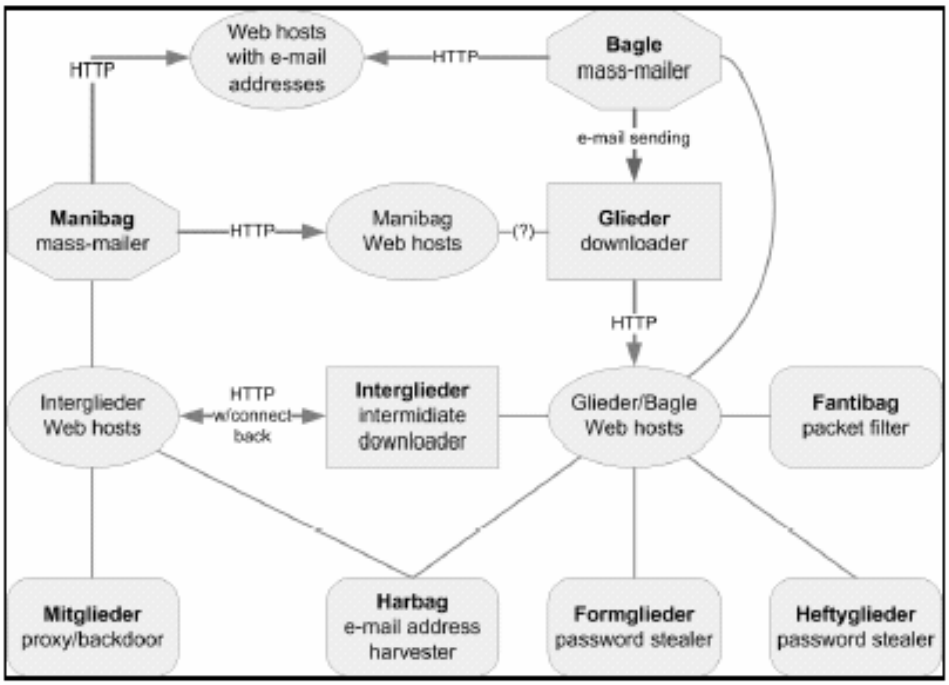

Figure 3. The Bagle Network