Journal of Information Assurance and Security 4 (2009) 69-83

Computer Worms: Architectures, Evasion Strategies, and Detection Mechanisms

Craig Smith, Ashraf Matrawy
Department of Systems and Computer Engineering, Carleton University
{csmith,amatrawy}@sce.carleton.ca

Stanley Chow, Bassem Abdelaziz
Alcatel-Lucent Bell Labs
{stanley.chow,bassem.abdelaziz}@alcatel-lucent.com

Abstract: This paper surveys the Internet worms-related literature and how stealthy worm behaviour can be discovered. Discussion is provided on the anatomy of worms, specifically covering the mechanisms by which worms spread, how they are detected, and how they may attempt to hide. The paper presents common detection mechanisms that we divide based on worm architecture properties. Namely, we summarize how worms can be detected at each of the following stages: target discovery, while they are being distributed, while being activated at the hosts, and when they run their payloads (where applicable). We also discuss some attack patterns for famous recent worms. The paper concludes with a discussion on current solutions (academic research, commercial products, and open-source tools) to detect worms and a comparative summary of these solutions/tools' capabilities.
Keywords: worms, evasion, detection, malware

I. Introduction

We survey the worm-related literature to highlight worm stealthy behaviour and methods of detecting it. A common question that arises when discussing worms is *What is the difference between a worm and a virus?*[57] Both are considered to be malware and can perform the same malicious actions. Viruses typically don't self-propagate, and rely on users to activate and transport the virus to a new destination. Worms are generally self-propagating, though the line between worms and viruses blurs when discussing mass-mailing viruses, since they self-propagate but usually rely on the receiving user to activate them [1]. The line also blurs when discussing newer all-encompassing malware which include features from viruses, worms, trojan horses and botnets, such as Storm [58]. In this paper, the term 'worm' is used to describe both worms and viruses, though the focus is on those which self-propagate across a network.

A. Host based vs Network-based Intrusion Detection Systems (IDS)

Most of the solutions presented in this report can be described as network-based or host-based. The distinction between the two is where each is implemented. Host-based solutions are implemented on the servers which run the services under scrutiny, and are useful for observing and changing the way the services operate when threats are detected. Host-based solutions can be implemented as modifications to the network stack, modifications to the operating system, modifications to the application, or even implemented by running the desired programs within an emulated environment.

Network-based IDS collect data directly from the network, and are installed at gateways between internal networks and the Internet. At these locations, the network-based IDS can analyze both incoming and outgoing traffic for malicious traffic. They can also be placed between subnetworks of larger organizations, such as between the networks of separate departments. Some existing host-based and network-based IDS are described in Sections VIII and IX.

B. Anomaly Detection vs Signature detection

Intrusion detection systems fall into two other general categories; anomaly detectors and signature detectors. The two categories fill different niches. Signature detectors, such as Snort, Bro and Shield, use pre-generated signatures to detect traffic or behaviour that is known to be malicious or undesirable. The signatures can be either man-made, or automatically generated, though in either case, an ideal signature should match only the malicious traffic and no legitimate traffic. Signature detectors can detect known attacks but are unable to detect new attacks. Since fast-spreading worms can infect most of the Internet before man-made signatures can be crafted [39], automated signature generating systems are needed. These signature-generating systems are typically called anomaly detectors.

Received December 20, 2008

1554-1010 $ 03.50 Dynamic Publishers, Inc.

Anomaly detectors detect unusual patterns, or anomalies, in network traffic or in the behaviour of a running program. The anomaly being detected depends strongly on which anomaly detector is being used. Examples of some current methods include detecting segments of executable code, unusual byte frequencies, unusual flags in packet headers, packet data being used to overwrite the return address within a program and infrequent sequences of bytes in network traffic. Once an anomaly is detected, most anomaly detectors can then generate a signature which can be used by signature based IDS. Existing anomaly and signature-based IDS are described in Sections VIII and IX.

The contributions of this paper can be summarized as follows: (a) surveying and analyzing different strategies for detecting worm behaviour at different stages of worm propogation with emphasis on the stealthy worm cases, and (b) comparing existing tools and solutions according to their capabilities of detecting different types of worm behaviour.

The rest of this paper is organized as follows: Sections II through V discuss detection of worms based on different architectural properties [1]. Section VI describes common worm behaviour, with examples of attack patterns. Section VII discusses techniques worms may take to avoid detection. Sections VIII and IX describe existing tools that are either sold commercially or developed academically to detect worms. Section X contains a grid which describes which tools are effective at detecting or blocking different aspects of worm propagation.

II. Detecting Target Discovery

A. Target discovery

The process of target discovery or scanning is the process where worms find new hosts to infect and is a characteristic behaviour that worms exhibit which can be detected and stopped at the network level. A host-based approach could also be used, however, detecting target discovery at the network level would yield more scans since it detects the traffic directed at multiple hosts. A detailed discussion on target discovery by Staniford et al. can be found in [57]. It discusses

- random scanning
- permutation scanning
- localized scanning
- hit-list scanning
- topological scanning
- metaserver scanning
- passive scanning

The simplest form of scanning is random scanning. The attacker selects a target at random, probes that target, and then continues the cycle by generating a new random target. Permutation scanning is an improvement on random scanning, allowing the attacker to avoid probing the same address multiple times and to coordinate the scanning of many infected hosts. Staniford et al. [57] suggest a simple implementation of permutation scanning for a 32-bit address space.

Another form of target discovery is localized scanning. An attacking host would preferentially scan local network addresses. Vogt [63] shows that worms using localized scanning can spread faster in the initial stages, but that once a large fraction of vulnerable hosts are infected, the infection process slows down. One of the attacker's advantages of scanning in this manner is that once a host behind a firewall is infected, it can directly infect other hosts without passing through the firewall, depending on network topology.

Worms can also use a pre-generated list of potential targets, known as a hit-list, to speed the rate of initial infections. The hit-list usually contains a list of addresses which are likely running vulnerable services. This hit-list can then be split up and distributed to newly infected hosts. A variant on this idea is to distribute an anti-hit list along with the worm which contains a list of networks to avoid scanning [63], [51]. These networks could simply be empty and thus be a waste of time to attempt to probe, or could be known to harbour network telescopes attempting to automatically generate worm signatures. After exhausting the hit-list, a worm could switch to permutation scanning and avoiding hosts in the anti-hit list.

Worms which employ topological scanning gather potential targets from the local machine. This includes the email addresses in a user's contact list, URLs in the user's browsing history. In a similar manner, the worms can query a metaserver to find potential targets, such as the services provided by Google, Gamespy or Netcraft, or by querying a peer-to-peer network or an instant messaging server for vulnerable peers.

A different and less common approach is for the worm to passively wait for incoming or outgoing connections and extract information from these connections to determine new targets [57],[1]. This form of scanning is much slower than the previous techniques but can be harder to detect by intrusion detection systems (IDS). Two examples are Gnuman [6] and CRClean [21] (described below).

B. Detection

The scanning of most worms can be detected by using anomaly detection. The idea is to put bounds on what is considered 'normal' traffic and trigger an alarm when those bounds are exceeded. Recent high-profile worms have tried to spread as quickly as possible. The anomaly in this case may be that the infected host is contacting many unique IP addresses in a short time span, or is receiving too many TCP RST packets, indicating many failed connection attempts. Multiple IDS' are compared in [11], where the IDS' attempt to identify scanning hosts.

Random, permutation and localized scanning may exhibit both anomalies since they can target the whole IP range and will not know whether the vulnerable service is running at each address. Hit-list, topological and metaserver scanning

are less likely to exhibit the TCP RST packets anomaly, since they have a greater likelihood of targetting a host running the desired service. Passive scanning is unlikely to show either anomaly, and may potentially only be detectable by means other than target discovery.

Honeypots are another way to detect scanning hosts. Honeypots are hosts to which should have no incoming traffic. When traffic is received by a honeypot, it means that either someone connected to the honeypot by mistake, or a worm is attempting to connect. Unless using a comprehensive anti-hit list, the random, permutation and localized scanning worms will likely send packets to a honeypot eventually. The larger the number of addresses feeding the honeypot, the more quickly scanning hosts will be discovered. However, a larger honeypot, depending on its behaviour, is also more likely to be added to an anti-hit list. Some honeypot systems can be set up to respond to traffic sent to an organization's unused IP addresses. An attacker scanning these organizations would see that an unnaturally high percentage of IP addresses contain hosts and infer that it is running a honeypot. Hit-list, topological, metaserver and passive worms are unlikely to be discovered by a honeypot because the hit-list will have been generated using likely-vulnerable hosts, and the metaserver, topological and passive worms will not include the honeypots since honeypots are unannounced and don't provide any useful publicly available services.

Another approach is to use network address space randomization [9]. The goal of this approach isn't to detect the worms, but rather reduce the usefulness of hitlists by changing the IP addresses of hosts periodically. Shorter DHCP leases and changing IP addresses would lead to fewer valid hosts in a hitlist after a period of time. With fewer hosts to infect, simulations in [9] show that if this approach is implemented by a small fraction of organizations, the propagation of a worm to 90% of it's vulnerable population can take twice as long as without address space randomization.

III. Detecting Worm Distribution

A. Distribution mechanisms

Three main mechanisms for worm distribution are identified by Weaver et al. [1]. They can be

- self-carried
- embedded, or
- use a secondary channel

Self-carried worms are fully transmitted during the initial communication with the target. Worms that rely on a second channel, such as the Blaster worm, send the infection in two stages. The worm first sends a small message which includes a small program which will download and run the rest of the worm. The third type is embedded worms, which transmit themselves within normal communications channels either by appending themselves to normal messages, or replacing normal messages.

B. Detection

Worm distribution can be detected with network-based IDS which can scan both incoming and outgoing traffic. Since worm distribution may occur at the same time as target discovery (such as with UDP-based worms like Slammer), some worms may be detected by the same mechanisms as explained above in Section II-B. Single-packet UDP worms can spread very quickly [37], and only warrant special attention because of that speed. They can be detected in a manner similar to other worms.

In addition to the previously described methods, the contents of the worm itself can be analyzed when the worm is transmitted over the network. A simple worm will send identical copies of itself across the network. If many packets are found to contain identical byte sequences, this may signal the presence of a new worm. EarlyBird [54] operates with this strategy and can automatically generate a signature for such worms. Encrypted or polymorphic worms would be able to bypass this detection method since there would be fewer byte sequences common between each variation of the worm. Section VII-B discusses encrypted and polymorphic worms in more detail.

Worms requiring a second channel have a higher chance of being detected since they initiate at least two connections. One problem with developing a worm which uses a second channel is that firewall rules may prevent the second channel communication entirely.

IV. Detecting Worm Activation

A. Activation mechanisms

Four methods by which worms are activated are discussed by Weaver et al. [1]. They can be

- human activated
- activated based on human activity
- activated by a scheduled processes, or
- self activated

Human activated worms need a human to manually execute the worm and are often referred to as viruses. Worms which are activated when a user clicks on an email, such as the Melissa virus, or which copy infected files onto a shared folder, such as the Nimda worm, fall into this human-activated category. The second worm activation method is human activity-based activated worms, which are activated by a user's actions which wouldn't normally be expected to execute a worm, such as via a user's login scripts, or when a CD or memory card is inserted into the computer.

Scheduled process activated worms are activated by a legitimate automated process which hasn't been properly secured, such as a legitimate program which automatically updates itself from an infected web server.

Self activating worms are the most worrisome and begin execution immediately after being transmitted to the target.

These worms generally exploit a vulnerability in a running application. Buffer overflow vulnerabilities are a common target, and there have been attempts to detect packets containing buffer overflows [71]. Code Red I and II are examples of self activating worms.

B. Detection

Since the activation occurs on the host, only host-based IDS will be able to detect the activation. Network-based IDS may detect anomalous traffic once the worm has already been activated. Host-based IDS which monitor system calls, implement taint tracking or detect buffer overflows can detect self-activated worms. IDS which monitor system calls, such as pH [56] and [52], detect the worm based on unusual sequences of system calls. Taint-based IDS such as TaintCheck [41], or Vigilante [20] mark incoming network data with a 'taint' as well as the results of any calculations which operate on the tainted data. These systems detect malicious behaviour when the return address to a function is overwritten by tainted data. Other methods of detecting buffer overflows include checking that a function's return address hasn't been changed, relying on a modified return address pointing to an invalid location, or pointing to invalid instructions. Vigilante uses the former and COVERS [32] uses both, and another system [16] combines taint analysis with anomaly detection. The previous techniques work for detecting self-activated worms, however may not work for the other three activation methods. The taint-based IDS and the ones relying on modified return addresses will not detect a worm if it is executed directly by the user. Host-based antivirus solutions are effective at detecting known worms and preventing them from infecting a host. Since antivirus solutions are often signature-based, they may not detect novel worms for which signatures have not yet been created.

V. Detecting Worm Payloads

A. Different forms of worm payloads

The payload of a worm refers to the behaviour or actions taken by the worm. Weaver et al. [1] discuss many types of payloads that a worm can contain, which are presented below.

Worms can allow an attacker to remotely control the infected host. In this case, the attacker can execute arbitrary code and can cause the infected host to take any desired action. This could include mounting a DoS attack against a target, collecting data from the infected host, such as keylogging, or erasing, modifying or encrypting files on the infected host.

A worm can cause the host to act as a mail relay or a web proxy. Mail relays are used by spammers to cloak the source of spam and web proxies would be used to cloak the source of undesirable websites, such as phishing sites.

Worms can also be used to cause physical damage such as reflashing a host's BIOS or swamping 911 services by dialling with a modem.

B. Detection

Detecting worm payloads (behaviours) can be done by actually running the worm, either in a simulated or instrumented environment, or by analyzing the instructions and system calls made by the worm. If running the worm produces behaviours that generate network traffic, that traffic can signify the presence of a worm, but will only be useful for signature generation if the worm attempts to spread. A system such as the double honeypot, described in section VIII, would detect this kind of network traffic.

VI. Attack Patterns

Identifying worm attack patterns would be similar to identifying the behaviour of a worm. There are endless permutations of how a worm can behave, limited only by the imagination of the author of the worm. In practice, the behaviour of the worm is limited to delaying for arbitrarily long periods of time and the activities listed in section V. To give a flavour of what worm behaviours have been seen in the past, a summary of some of the more interesting worms in recent history are detailed below.

- Code Red [38] used a buffer overflow vulnerability to infect Microsoft IIS web servers. Code Red would scan and infect other hosts until the 20^{th} day of the month and then send a DoS attack to the whitehouse website until the 28^{th} and then become dormant for the remainder of the month. Some of the infected web servers added the phrase 'Hacked by Chinese' to the web pages they served. This process would repeat every month, however it did not cause very much damage. Code Red was a memory-resident worm, so it did not persist across reboots.

- Code Red II [38] used the same buffer overflow vulnerability as Code Red, but was otherwise completely different. It first determined if Code Red II was already installed, and if not it installed a backdoor, went dormant for a day and then rebooted the machine. It then began to spread. Installing the backdoor allowed a remote user to execute arbitrary code at a later date.

- Nimda [34] used four exploits to infect web servers running IIS, IE web browsers and Office 2000 programs. It infected IIS web servers, which then infected visitors who use IE. It also copied itself to network drives, shared the computer's folders, and created a guest account with Administrator privileges. It attached itself to explorer.exe to hide itself. It emailed itself to email addresses in the user's contact list. It was self-modifying, so hashes wouldn't identify it. Similar to the Code Red II worm, the Nimda worm allowed remote access to the

infected host, as well as decreasing the security of that host by sharing the C drive.

- Slammer [37] was a fast-spreading worm that contained no malicious payload. It was fully contained in a 404-byte UDP packet and used a crippled implementation of random scanning. It took advantage of a buffer-overflow vulnerability in SQL Server.

- Slapper [10] spread by exploiting a vulnerability in the OpenSSL implementation used by the Apache web server. It scanned for targets by randomly choosing a network, and then sequentially scanning each IP in that network. The Slapper worm maintained a P2P network of other infected hosts and could be used to anonymously forward commands from the controller to any infected machine. The commands could cause the infected hosts to participate in a denial of service attack against arbitrary targets.

- The Witty worm targeted an overflow vulnerability in eEye's Internet Security System's [3] protocol analysis module for ICQ communications. The worm had a large initial population, which suggests the use of a hit-list and reached saturation after infecting 12000 hosts in under 45 minutes. It used random scanning, and alternated between scanning for new targets and overwriting random sections of the local hard drive.

- Blaster [5][13] exploited a buffer overflow vulnerability in the RPC implementations of Windows XP and 2000. Blaster was uploaded to the target in two stages. The first stage transmitted itself via the RPC vulnerability, which then retrieved and executed the rest of the worm. Blaster was designed to send a SYN flood to windowsupdate.com on certain dates.

- Conficker [48] was another recent worm. Variants of Conficker would install themselves onto flash drives and network shares, but originally used a vulnerability in RPC on Windows XP and 2000. Conficker could update itself by locating updates at regular intervals, or the attacker could push updates to the infected hosts.

- Gnuman [6] joined a Gnutella network as a normal node and propagates by tricking users into downloading and executing it. The deception was done by responding to any search with an infected executable file with the same filename as the search phrase. Gnuman did nothing else.

- CRclean (Code-Red Clean) [21] was a proof-of concept worm that was never released. It was designed to wait for attacks from hosts infected with Code Red II and then infect the attacking host, disable Code Red II, and filter out subsequent Code Red II infection attempts.

VII. Evasion Strategies

In order to spread as widely as possible, worms can either attempt to propagate quickly in an attempt to outrun the automated signature generators, or by avoiding the automated signature generators entirely by spreading in a covert manner. This section discusses some evasion strategies that worms can take, such as scanning slowly, using polymorphism or encryption, blending with normal traffic, and different attempts the worms can make to mislead signature generators.

A. Slow Scanning

Some strategies for detecting worms rely on worms revealing themselves by quickly scanning many addresses in a short time. Solutions such as EarlyBird [54] rely on gathering multiple pieces of similar network traffic before sounding an alarm. To reduce the amount of processing and/or the amount of memory needed, EarlyBird and other approaches set a threshold at which infrequent or uncommon packets are discarded. If a worm spreads at a slow enough rate such that it is always discarded, the automated system will never generate a signature and the worm can continue to spread.
Current research doesn't seem to be focused on detecting a worm based on its speed, but rather by making an anomaly detection system so accurate that it can reliably detect a malicious packet the first time it is received. An advanced network-based IDS such as Anagram [68] would be able to detect the anomalous packets, and any host-based IDS such as TaintCheck [41] would be able to detect harmful behaviour in a running service.

B. Polymorphic Worms and Encrypted worms [29]

To evade signature-based IDS, worms may try to modify themselves using either polymorphism, encryption or both. The idea is that the modified worm will no longer match the existing signature and so it will pass through the IDS undetected. Since the polymorphic or encrypted worm must run to properly infect a host, a host-based IDS will be able to detect the worm [41]. Any other constants that are required in a packet to successfully exploit a vulnerability can be detected by most IDS, so a polymorphic and encrypted worm would not be a useful strategy if the exploit is easily detectable. Such a worm would be detected by either host- or network-based IDS.
A polymorphic worm will modify its code by rearranging functional blocks of code, by replacing instructions with functionally equivalent instructions, or by inserting groups of instructions which do not change the behaviour of the program, but rather act as padding [29]. Since the code for the worm can change each time it is transmitted, the code of the worm will not match any worm signatures. Polygraph [43](VIII) was designed to detect polymorphic worms. Instead of looking for one common byte sequence among packets, it looks for smaller sequences of common bytes, which

are then used as part of a new signature.
An encrypted worm will generate a random encryption key, encrypt itself, and then transmit that encrypted version along with the key and a short decryptor program. The decryptor program itself can be polymorphic so that the whole executable part of the worm can change. Similarly to the polymorphic worms, the encrypted and polymorphic portion of the worm will not be detectable by signature-based IDS.
Assuming an encrypted and polymorphic worm which uses an exploit that has few detectable constants, a host-based IDS such as Vigilante [20](VIII) or TaintCheck would be needed to detect it, or a network-based IDS which looks for payload anomalies, such as PAYL [69] and Anagram. Both Vigilante and TaintCheck detect when code at incorrect addresses are being run, and can trace those addresses back to a location in received packets. The data at those locations can be used to generate signatures. PAYL determines whether a packet is suspicious based on its byte distribution. After receiving a few more similar packets, it can generate a signature based on the longest common substring, or longest common subsequence of the similar packets. This common substring or subsequence would likely be the constant parts of the worm.

C. Blending/Mimicry

Blending attacks, or mimicry attacks occur when a worm attempts to pass through an IDS without detection by changing itself or its behaviour to appear similar to normal traffic or normal behaviour. Mimicry approaches, as described in [29] and [24], can be used to evade network-based IDS such as PAYL. Both techniques involve giving a worm a learning step before attempting to infect other hosts. The worm would analyze local network traffic to generate a normal traffic profile, and then modify its outgoing traffic to match that normal profile. The IDS PAYL and NETAD [35] each detect changes in the byte frequency distributions of packets. To avoid these IDS, a worm could append filler bytes at the end of packets in order to make the byte frequency distribution match the normal profile.
Host-based IDS which instrument running processes can detect worms which send mimicked traffic since the behaviour of the executing worm would be the same. Network-based IDS such as Anagram [68] can also be used. Anagram was designed specifically to defeat mimicry attacks. Since previous mimicry strategies generate packets which match the overall statistics of normal network traffic, Anagram subdivides the packets into separate partitions. Normal network traffic statistics are determined for each partition. Since the worm will not know where the bounds of the partitions are, it will not be able to effectively create a 'normal' looking packet.
Mimicry can also be used at the host level. Some host-based IDS, such as pH [56], monitor system calls used by a program. D. Wagner and P. Soto [65] describe how a worm could call the system calls in a manner that mimics the pattern of normal programs. They describe how pH keeps track of sequences of system calls to determine if a program is suspicious, and then describe how some system calls calls can be used as no-ops. A worm can intersperse its regular system calls with the no-op system calls in order to avoid IDS which monitor system calls. J. Griffin, et al. [25] propose a host-based system which analyzes the system calls of a program to automatically determine if any system call sequence exists in the program which could result in a security breach. Since the tool described in [25] analyses the system calls of an executable, it will not detect buffer overflow attacks.

D. Misleading signature generators

To avoid detection, worms can attempt to deceive the IDSs which try to generate new signatures. The effectiveness of each approach depends on the IDS that is implemented. The approaches include misleading the IDS by

- causing it to generate useless signatures
- sending 'allergy attacks' or other learning attacks
- splitting up packets, or
- overloading the IDS

Perdisci et al. [47] discuss how IDS such as Polygraph can be fooled into generating signatures which are of no use and allow worms to avoid being detected. This is one type of learning attack, other such learning attacks are described in [14]. In order to fool signature generators, the normal infection attempt is sent to the target host along with a second (or third) 'fake' infection attempt. The fake attempts are crafted such that they closely match the real infection attempt in many of the 'polymorphic' areas of the worm, but not in the invariant areas of the worm. This approach causes the IDS to notice the common bytes in the polymorphic areas and generate a signature which matches that specific variation of the worm, but will fail to identify any other variations of that same worm. The technique discussed in [47] showed that Polygraph could be evaded 85% of the time on average. Part of this approach relies on knowing or learning some common sequences of bytes in normal traffic. This technique can be used to avoid some network-based IDS, but will still be detected by host-based IDS, since the unexpected execution of the worm can be detected.
While the previous learning attack causes IDS to generate useless signatures, another learning attack, called a causative integrity attack [14], can be used to prevent any signature from being generated in the first place. This attack is discussed briefly in [68] and in more detail in [53]. The attacks can generally be done in two scenarios: where the original training data contains unkown malicious traffic, and where an IDS is set to continuously learn from incoming traffic. In the latter case, a worm may be able to train the IDS to recognize data from the worm payload as normal traffic. By sending traffic which successively changes from normal to worm traffic, an IDS may be gradually convinced that the worm traffic is legitimate and so it will not generate a sig-

nature. Another learning attack, called an availability attack, or 'allergy attack', could cause the IDS to erroneously block innocuous traffic and lead to the network administrator disabling the IDS. M. Barreno et al. [14] describe how many of these learning attacks can be prevented, while S. Chung [19] describes how allergy attacks are possible in recent IDS.
They suggest biasing a decision if the data is particularly noisy (regularization), changing the boundary at which traffic is classified as malicious or not (randomization) and reducing the information available to the attacker (information hiding).
Since TCP traffic can be fragmented into smaller packets, an attacker may be able to split up large anomalous worm packets into many small packets which raise no alarms. Many IDS such as EarlyBird and Snort take this into consideration and attempt to reassemble fragmented packets in order to detect these evasion attempts.
An attacker could also mount a denial of service attack against an IDS in an attempt to overload the IDS such that it can no longer detect worms. Host-based approaches which instrument a running program, such as TaintCheck, are particularly at risk since the running program already runs much more slowly than normal. IDS designers reduce the likelihood of an overloaded IDS by pre-filtering the incoming traffic so that known-good traffic doesn't unduly burden the IDS and by using data structures that require very little memory. Snot is a program which was designed to generate large amounts of malicious-seeming traffic. It generates traffic based on Snort rules, so is likely to generate large numbers of alerts on a Snort IDS. A similar approach is described in [45].

VIII. Academic Approaches

These approaches are described in papers and either attempt to identify, slow or stop the spread of worms. Commercial implementations for these approaches, however, are not readily available or not publicized. The network-based intrusion detection systems are listed at the beginning of the list while host based systems appear at the end.

- EarlyBird [54] generates Bro and Snort signatures automatically based on how many sources and destinations are sending similar packets. The idea is that when many copies of a worm send copies of themselves to infect other hosts, there will be many hosts sending copies of the worm and many hosts receiving them. EarlyBird flags packets as suspicious if multiple senders send, and multiple receivers receive similar traffic. EarlyBird keeps a small amount of state to keep track of common sequences of bytes and generates a signature when many IP addresses are sending and receiving these common sequences of bytes. EarlyBird is a network-level implementation and was measured to run at line speeds of 200Mbps, but can scale higher.

- Autograph [28] is a distributed system for automatically generating worm signatures for Bro and Snort. It is intended to be installed at the DMZ of edge networks. The suggested sample implementation classifies traffic as suspicious if a RST packet was generated inside the network, so this will only work if the connection is symmetric. The idea is that worms will scan many hosts which aren't running the vulnerable service, and will generate TCP packets with the RST bit in response to the worm's scanning. Autograph is designed to share suspicious sources of traffic with other Autograph hosts.

- 'Throttling viruses': A small network or system-level approach to reducing the speed at which worms can propagate. The idea is to keep a small cache of the most recent connections that were initiated. Each new connection which connects to a host that isn't in the cache gets queued, and the queue is processed at X (˜=1 or 2) connections per second. Since worms will generally try to spread by making connection as quickly as possible, most of the new connections will be queued and processed slowly. [72]

- Polygraph [43]: A network-level approach to automatically generating signatures for polymorphic worms. The idea is to detect multiple invariant byte sequences instead of a single large sequence. Polygraph is trained on some known-good traffic and some likely-malicious traffic before being deployed.

- Threshold Random Walk (TRW): This method attempts to discover worms by their scanning behaviour. It detects scanning IP addresses based on how many destinations are scanned and how many of those connection attempts are rejected or unanswered. The idea is that legitimate clients are more likely to be looking up servers using DNS, and will likely succeed in connecting to the desired server, while scanners are more likely to be choosing targets randomly, which may not be running the expected server software. Whether a host is scanning or not can be determined after a host makes 5 connection attempts. [27]

- NETAD [35] is a network-based IDS. It takes the first 48 bytes of the first few packets (which contains mainly IP and TCP header information) between a server and client. NETAD determines whether a packet is anomalous or not based on how 'novel' a byte in one of the 48 first bytes is. The more 'novel' bytes the more anomalous the packet appears. PHAD, ALAD and LERAD are briefly discussed in the same paper and seem similar.

- SigFree [71] is a host-based IDS which generates signatures in traffic that contains long sequences of executable code. A common buffer overflow technique is to include a NOP sled (or equivalent instructions) along with the exploit code to save the attacker from needing

to know the exact address where the exploit code will be located. Since the sled is by definition executable code, SigFree can detect packets with long sequences of executable code and flag them as suspicious. SigFree is a server- or client-side proxy through which the desired communications are passed. SigFree seems to be an extension of Abstract Payload Execution [61], which is implemented as an Apache module.

- PAYL (PAYLoad Anomaly Detection) [69] [67] is a network-based IDS which can generate signatures for malicious traffic. PAYL is initially trained with normal traffic, which gives PAYL the chance to learn the average distribution of bytes within packets. PAYL learns the distributions for each combination of port and packet length. A distance metric is used to determine how close a packet is to the normal data. If the distance is above a specified threshold, the packet is marked as suspicious. PAYL also correlates anomalous incoming packets with anomalous outgoing packets, for example, if a suspicious packet arrives on port *i*, and the recipient then sends out a similar suspicious packet destined for port *i* on another host, it is likely the result of the host being infected. This behaviour allows PAYL to detect the very early stages of worm propagation. PAYL can then generate a signature based on the longest common substring (LCS) or longest common subsequence (LCseq), which can be used in other IDS. There are multiple variations of PAYL's strategy, one of which [60] attempts to speed up IDS processing by reducing the amount of packet data passed to PAYL.

- Anagram [68] is a recent network-based IDS which is designed to detect mimicry attacks and generate signatures which can be used by others. Anagram uses two Bloom filters to keep track of normal and bad traffic. The 'normal traffic' Bloom filter is trained using normal traffic and the 'bad traffic' Bloom filter can be generated from previously generated signatures which match undesirable traffic.

  Anagram searches incoming packets for n-grams (byte sequences with length n) which either exist or do not exist in its Bloom filters. If either many n-grams from the 'bad traffic', or very few n-grams from the 'normal traffic' filter exist in the packet, that packet is flagged as malicious. The authors suggest that Anagram could use feedback from a heavily instrumented 'shadow server' to improve its filters. The authors expect Anagram to be able to handle traffic at 100Mbps while using around 20MB of memory.

- The double honeypot [59] is both a network- and host-based IDS which consists of an inbound honeypot and an outbound honeypot. The inbound honeypot is a high-interaction honeypot, meaning that it runs an operating system and one or more programs. The inbound honeypot is configured such that, by default, it never makes outgoing connections. Any outgoing connections are evidence that a worm has taken over the honeypot. Any outgoing connections are redirected to the outgoing honeypot for analysis. The paper which describes the double honeypot also describes two methods for generating signatures for polymorphic worms.

- TaintCheck [41] is a host-based IDS which can generate signatures to identify malicious traffic. It does so by running the program of interest in an emulated environment and by considering any incoming data as 'tainted'. Any calculations derived from that tainted data are also considered tainted. When tainted data is used to determine the next executable instruction, TaintCheck will consider this malicious and will generate a signature. Programs run 1.5 to 30 times slower when TaintCheck is used.

- Vigilante [20] is a host-based IDS which generates a signature that can be verified by other hosts without requiring trust between hosts. An instrumented version of the desired service is run inside a virtual machine such that it will trigger alerts either when taint analysis has determined that tainted instructions are being executed, or when code is being executed from memory pages marked non-executable. Such an alert would cause Vigilante to generate a signature (or SCA, self-certifying alert) describing the offset in the received message which caused the alert. Vigilante is designed to run as a honeypot with low traffic. Minos [22] is another host-based IDS which operates in a similar manner to Vigilante.

- Shield [66] is a host-based intrusion prevention system which can prevent remote exploits by modifying or stopping malicious packets before they reach a vulnerable service. Shield uses signatures to detect the malicious packets. Since applying patches usually disrupts the running service, and patches may not be completely reliable, Shield can be used to prevent known exploits without having to restart the running service, allowing the administrator to decide when to apply software patches. Each signature is a small Shield-specific program which modifies the malicious packets.

- Vulnerability-specific execution-based filtering (VSEF): VSEF [42] is a host-based IDS which can 'harden' a program against a specific vulnerability. The patching works for programs where the source code isn't available and incurs just a small overhead. To operate, VSEF requires an execution trace of the program receiving the traffic which included the exploit. This trace is obtained by a fully-instrumented program somewhere else on the network, for example, using TaintCheck. Once VSEF has the trace, it can generate a VSEF filter which can then be sent to other

servers running the same program. VSEF can then 'harden' the program by adding code which filters out the exploit. Hardened programs run between 3 to 14% slower when VSEF filters are installed.

- The shadow honeypot [8] architecture is a host- and network-based IDS. It contains multiple anomaly detectors, such as Abstract Payload Execution (similar to SigFree) and EarlyBird. These IDS determine whether traffic is suspicious, and if it is, the traffic is redirected to a second 'shadow' server which runs an instrumented version of the server software. If the shadow server detects an attack, it reverts any changes it made and notifies a filtering component which filters out similar traffic before it arrives at the honeypot. Implementations for both client and server software were created.

- pH [56] (process homeostasis) is a host-based IDS which detects uncommon sequences of system calls. pH collects sequences of system calls during its training period, and stores subsequences of six consecutive system calls in a database. A program which generates a sequence that does not appear in the database is considered suspicious and will either be delayed or aborted. Many similar host based IDS are referenced in [65].

- COVERS [32] is a host-based IDS which can generate signatures. COVERS identifies attacks by detecting buffer overflows, by using address-space randomization, and instruction set randomization. It then correlates the attack with the packets involved in the attack, identifies the sections of the packets containing the attack and then generates a signature including this information. COVERS can be used with programs for which the source code is not available and runs with a low overhead of under 10%. This technique will not be able to generate specific signatures if the traffic is encoded or encrypted.

- Other promising IDS techniques include POSEIDON [15], a PAYL-based IDS, Argos [50], a taint-based IDS, Packet Vaccine [70], a host-base IDS, Sting [40], a taint-based IDS, SweetBait [49], a honeypot-based IDS, NetBait [18], a honeypot-based IDS, Prospector [55], taint-based honeypot IDS, WormTerminator [17], a polymorphic worm detector IDS, Hamsa [31], a polymorphic worm detector IDS, and Nemean [74], a signature generator IDS.

IX. Available Tools

These are tools that are available and can be purchased or downloaded immediately and put into production. They fall into two categories: honeypots and intrusion detection and prevention systems (IDS). The IDS can be either network-based or host-based. Network-based IDS detect malicious traffic and can take action before that traffic reaches the hosts in the network. Host-based IDS can detect malicious traffic once it has reached a host, and can analyze the contents of encrypted packets and detect when administrative policies are changed. Host-based IDS can be disabled if the host is compromised. The commercial products listed below were chosen because they represent the offerings from well-known companies or are frequently referred to. The descriptions of these commercial products are based on marketing material from each product's website.

- LaBrea [33] is A honeypot-like project which attempts to slow scanning worms. It responds to pings and responds to SYN packets with SYN-ACK packets, and does nothing else. It relies on the attacker waiting for the TCP connection to time out before continuing to scan. It can be set up to respond for unused IP addresses.

- Honeycomb [30] is a honeypot project which automatically creates worm signatures based on the traffic that is captured in the honeypot. Since the honeypot doesn't advertise any services, any traffic directed to the honeypot can be considered to be malicious. It uses a longest-common substring approach to find similarities in packet payloads. Other honeypot systems include HoneyTank [62], HoneyStat [23], Collapsar [26], iSink [73], Potemkin [64], nepenthes [12]. A summary of some of these approaches can be found in [44].

- Bro [46] is an open-source network based IDS which can detect and take action against malicious traffic. It can generate alerts or execute programs which take the desired action, such as terminating an existing connection or blocking future traffic from a hostile host. Bro uses rules describing restricted activities, policies for what activities should generate alerts and signatures to determine which action it should take.

  Bro is designed to provide real-time notification and to be extensible and resistant to attack. It is designed to do so in a high-speed network without dropping packets. Bro relies on libpcap to get packets from the network. Bro then generates events from the packets, such as connection_established or connection_attempted, and processes these events using the specified policies or scripts.

  Bro's signatures are based on regular expressions, which can be generated by automated tools.

  To resist attacks against the IDS, attacks are split into three categories: overload, crash and subterfuge attacks. Overload attacks attempt to overburden the IDS such that it fails to keep up with incoming packets. Overloading Bro is difficult since the attacker will not know when Bro becomes overloaded, and will also not know what policy scripts are being run. Crash attacks aim to disable the IDS, either by exploiting a vulnerability or by exhausting resources. Bro resists these by running a

watchdog process which terminates Bro when it detects when Bro is executing too slowly. Whenever Bro is terminated, either by the watchdog process or from a crash, tcpdump is started to continue logging any packets.

- Snort [4] is an open-source network based IDS similar to Bro. The detection rules to detect malicious traffic can be specified either as regular expressions or using Snort's built-in rule description language. The actions Snort can take in response to malicious traffic is limited to logging, alerting and changing iptable's rules to block traffic. Though less powerful than Bro's ability to run arbitrary commands, Snort's available actions provide the needed functionality of an IDS. Depending on the speed of the machine and Snort's configuration, Snort can process traffic at gigabit speeds.

- The Cisco Security Agent [2] is a host-based IDS. Versions of the client software are available for Windows, Solaris and Red Hat, in both server and desktop versions. The management software which coordinates with all of the agents handles up to 100,000 agents per management host. The software can add information to outgoing packets which can be used by Cisco routers for QoS purposes. Cisco's IPS 4200 systems are network-based IDS which operate at up to 1Gbps.

- TippingPoint [7] is a division of 3COM and offers a range of network-based IDS. It uses customized ASICs to scan traffic at up to 3 Gbps. It can detect zero-day worms and also uses signatures to detect previous worms. The signatures are provided by TippingPoint's 'Digital Vaccine' service. TippingPoint's 5000E device passed ICSA Labs' intrusion prevention system evaluation [36].

- Host-based firewalls: Popular firewall solutions include ZoneAlarm, Windows Firewall, BlackICE products, Norton Personal Firewall, and Comodo Personal Firewall. There are many others, and each may scan incoming and outgoing traffic for malicious packets and/or prevent unexpected incoming and outgoing connections.

- Host-based anti-virus: Popular antivirus solutions include Grisoft's AVG, Norton Antivirus, Kaspersky Anti-virus, ALWIL Software's Avast, and CA Anti-virus. There are many others, and each will usually provide services such as 'resident protection', periodic scanning and on-demand scanning for files, running programs, email attachments, instant messaging programs, etc. These products identify viruses and worms either via signature matching or by using heuristics. Since these are installed on the host, they can potentially be disabled by worms and viruses.

X. A Comparison

The grid below contains a list of products or approaches and lists whether each can detect or prevent existing worms, new worms, slow scanning worms, polymorphic or encrypted worms and worms using mimicry techniques. The 'training required' column specifies whether the product needs to be trained on sample data before the system can be used. The 'requires multiple samples' column specifies whether the system detects a worm only after it receives multiple samples of that worm. The 'speed' column contains information about how the system performs. The rate of network traffic the system can handle is specified when available, otherwise 'fast' or 'slow' is included. The products are used within small to large enterprises. Existing papers do not cover the behaviour of products at the ISP level.

XI. Concluding Remarks

This paper is an attempt to give a complete picture of the anatomy of computer worms, of how worms' behaviour can be stealthy, and of how to detect that behaviour. We discussed a number of issues that range from detection at different infection stages to evasion and detection strategies. Also a number of tools and research approaches have been discussed.

Despite the significant efforts by the research and development communities, the situation is far from perfect. Solutions exist to thwart the worms, but aren't necessarily practical for small businesses to implement. As a matter of fact, a new generation of worms is already spreading and starting to accumulate bots by the millions while the tradition signature-based tools are having trouble keeping up.

The change in motivation (from fame to cash) appears to be driving innovations in malware. We expect the monetary rewards to grow. The research and development communities need to be prepare for more innovative attacks and avenues that will attempt to monetize the infected machines.

Acknowledgment

This work is part of a project funded by Ontario Centers of Excellence and Alcatel-Lucent Canada.

green = good, red = bad, yellow = in between

	Existing Worms	Novel Worms	Slow Scanning	Polymorphic / Encrypted	Mimicry	Training Required	Requires Multiple Samples	Speed
Host Firewall Incoming and outgoing connections are never established.	Yes	Yes	Yes	Yes	Yes	No	No	Fast
Host Antivirus	Yes	No	n/a	Yes	n/a	No	No	Fast
Honeycomb	Yes	Yes	No	No	Yes(3)	No	Yes	Unk.
Bro/Snort	Yes	No	Unknown	Unknown	Unknown	No	No	Varies
Cisco(1)	Yes	Yes	Unknown	Unknown	Unknown	Unknown	Unknown	2Gbps
TippingPoint(1)	Yes	Yes	Unknown	Unknown	Unknown	Unknown	Unknown	3Gbps
EarlyBird	Yes	Yes	No	No	No	No	Yes	200+ Mbps
Autograph	Yes	Yes	No	No	No	Yes	Yes	Unk.
Polygraph	Yes	Yes	No	Yes	No	Maybe another system pre-classifies incoming traffic, could use pre-training.	Yes	Unk.
TRW	Yes	Yes	No	Yes	Yes Contents of packets not analyzed, so content mimicry doesn't affect detection.	No	Yes	Fast
NETAD	Some	Some	Some	Some(2)	No	Yes	No	Unk.
SigFree SigFree will likely perform poorly if it is analyzing traffic which normally transfers executable files.	Yes	Yes	Yes	Some	No	No	No	100 Mbps
PAYL	Yes	Yes	Yes	Yes(2)	No	Yes	No	Fast
Anagram	Yes	Yes	Yes	Yes(2)	Yes	Yes	No	Fast
TaintCheck	Yes	Yes	Yes	Yes	Yes	No	No	Slow
Vigilante	Yes	Yes	Yes	Yes	Yes(3)	No	No	Slow
Shield	Yes Requires a human-generated signature.	No	Yes	Yes Shield is targeted to the vulnerability not the worm.	Yes	No	No	350 Mbps
Double Honeypot	Yes	Yes	Yes	Some	Yes(3)	No	Yes	Unk.
VSEF	Yes	Yes	Yes	Yes	Yes	No	No	Fast
Shadow Honeypot	Yes	Yes	Yes	Yes	Maybe(4)	Maybe(4)	Maybe(4)	1Gbps
pH	Yes(5)	Yes(5)	Yes(5)	Yes(5)	No	Yes	No	Fast
COVERS	Yes	Yes	Yes	Yes	Yes	No	No	Fast

1. Claimed in marketing material.
2. Not likely if normal traffic contains large amount of encrypted traffic.
3. Since this is a honeypot, any traffic arriving is suspicious. Mimicry will not work in this case.
4. Depends on the anomaly detectors being used.
5. Yes, unless an attacker specifically attempts to avoid this type of detection.

References

[1] N. Weaver and V. Paxson and S. Staniford, and R. Cunningham. A Taxonomy of Computer Worms. In *The First ACM Workshop on Rapid Malcode (WORM)*, 2003. 11.

[2] Cisco Systems Inc's Cisco Security Agent website. Available at http://www.cisco.com/en/US/products/sw/secursw/ps5057/index.html, viewed May, 2009.

[3] eEye Digital Security's SecureIIS. Available at http://www.eeye.com/html/products/secureiis/index.html, viewed May, 2009.

[4] Snort User Manual 2.8.4. Available from http://www.snort.org/docs/snort_htmanuals/htmanual_284/, viewed May, 2009.

[5] Symantec summary of W32.Blaster.Worm. Available at http://securityresponse.symantec.com/avcenter/venc/data/w32.blaster.worm.html, viewed May, 2009.

[6] Symantec summary of W32.Gnuman.Worm. Available at http://www.symantec.com/security_response/writeup.jsp?docid=2001-022710-3046-99, viewed May, 2009.

[7] 3COM's TippingPoint IDS website. http://www.tippingpoint.com/products_ips.html, viewed May, 2009.

[8] K. G. Anagnostakis, S. Sidiroglou, P. Akritidis, K. Xinidis, E. Markatos, and A. D. Keromytis. Detecting targeted attacks using shadow honeypots. In *Proceedings of the 14th conference on USENIX Security Symposium*, pages 129–144, 2005.

[9] S. Antonatos, P. Akritidis, E. P. Markatos, and K. G. Anagnostakis. Defending against hitlist worms using network address space randomization. In *WORM '05: Proceedings of the 2005 ACM workshop on Rapid malcode*, pages 30–40, New York, NY, USA, 2005. ACM Press.

[10] Iván Arce and Elias Levy. An Analysis of the Slapper Worm. *IEEE Security and Privacy*, 1(1):82–87, 2003.

[11] Ayesha Binte Ashfaq, Maria Joseph Robert, Asma Mumtaz, Muhammad Qasim Ali, Ali Sajjad, and Syed Ali Khayam. A Comparative Evaluation of Anomaly Detectors Under Portscan Attacks. In *Recent Advances in Intrusion Detection: 11th International Symposium, RAID 2008, Cambridge, Massachussets, U.S.A.*, pages 351–371, 2008.

[12] Paul Baecher, Markus Koetter, Thorsten Holz, Maximillian Dornseif, and Felix C. Freiling. The Nepenthes Platform: An Efficient Approach to Collect Malware. In *Recent Advances in Intrusion Detection, 9th International Symposium, RAID 2006, Hamburg, Germany*, pages 165–184, 2006.

[13] Michael Bailey, Evan Cooke, Farnam Jahanian, David Watson, and Jose Nazario. The Blaster Worm: Then and Now. *IEEE Security and Privacy*, 03(4):26–31, 2005.

[14] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. D. Tygar. Can machine learning be secure? In *ASIACCS '06: Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 16–25, New York, NY, USA, 2006. ACM Press.

[15] Damiano Bolzoni, Sandro Etalle, Pieter Hartel, and Emmanuele Zambon. POSEIDON: a 2-tier Anomaly-based Network Intrusion Detection System. In *IWIA '06: Proceedings of the Fourth IEEE International Workshop on Information Assurance (IWIA'06)*, pages 144–156, Washington, DC, USA, 2006. IEEE Computer Society.

[16] Lorenzo Cavallaro and R. Sekar. Anomalous Taint Detection. Technical Report SECLAB08-06, Stony Brook University, 2008.

[17] Songqing Chen, Xinyuan Wang, Lei Liu, and Xinwen Zhang. WormTerminator: an effective containment of unknown and polymorphic fast spreading worms. In *ANCS '06: Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems*, pages 173–182, New York, NY, USA, 2006. ACM Press.

[18] B. N. Chun, J. Lee, and H. Weatherspoon. Netbait: a Distributed Worm Detection Service. Technical Report IRB-TR-03-033, Intel Research Berkeley, September 2003.

[19] Simon P. Chung and Aloysius K. Mok. Advanced Allergy Attacks: Does a Corpus Really Help? In *Recent Advances in Intrusion Detection, 10th International Symposium, RAID 2007, Queensland, Australia*, pages 236–255, 2007.

[20] Manuel Costa, Jon Crowcroft, Miguel Castro, Antony Rowstron, Lidong Zhou, Lintao Zhang, and Paul Barham. Vigilante: end-to-end containment of internet worms. In *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 133–147, New York, NY, USA, 2005. ACM Press.

[21] S. Coursen. Good viruses have a future. Available at http://www.securityfocus.com/columnists/23, viewed May, 2009.

[22] Jedidiah R. Crandall, Shyhtsun Felix Wu, and Frederic T. Chong. Experiences Using Minos as a Tool for Capturing and Analyzing Novel Worms for Unknown Vulnerabilities. In *Proceedings of Detection of Intrusions and Malware, and Vulnerability Assessment, Second International Conference (DIMVA)*, pages 32–50, 2005.

[23] David Dagon, Xinzhou Qin, Guofei Gu, Wenke Lee, Julian B. Grizzard, John G. Levine, and Henry L. Owen. HoneyStat: Local Worm Detection Using Honeypots. In *Recent Advances in Intrusion Detection: 7th International Symposium, RAID 2004, Sophia Antipolis, France*, pages 39–58, 2004.

[24] Prahlad Fogla and Wenke Lee. Evading network anomaly detection systems: formal reasoning and practical techniques. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 59–68, New York, NY, USA, 2006. ACM Press.

[25] Jonathon T. Giffin, Somesh Jha, and Barton P. Miller. Automated Discovery of Mimicry Attacks. In *Recent Advances in Intrusion Detection, 9th International Symposium, RAID 2006, Hamburg, Germany*, pages 41–60, 2006.

[26] Xuxian Jiang and Dongyan Xu. Collapsar: a vm-based architecture for network attack detention center. In *Proceedings of the 13th conference on USENIX Security Symposium*, pages 15–28, 2004.

[27] Jaeyeon Jung, Vern Paxson, Arthur W. Berger, and Hari Balakrishnan. Fast Portscan Detection Using Sequential Hypothesis Testing. In *IEEE Symposium on Security and Privacy 2004*, pages 212–225, Oakland, CA, May 2004.

[28] Hyang-Ah Kim and Brad Karp. Autograph: Toward Automated, Distributed Worm Signature Detection. In *USENIX Security Symposium*, pages 271–286, 2004.

[29] Oleg Kolesnikov and Wenke Lee. Advanced Polymorphic Worms: Evading IDS by Blending in with Normal Traffic. Technical Report GIT-CC-05-09, College of Computing, Georgia Tech, 2005.

[30] Christian Kreibich and Jon Crowcroft. Honeycomb - Creating Intrusion Detection Signatures Using Honeypots. In *Proceedings of the Second Workshop on Hot Topics in Networks (Hotnets II)*, Boston, November 2003.

[31] Zhichun Li, Manan Sanghi, Yan Chen, Ming-Yang Kao, and Brian Chavez. Hamsa: Fast Signature Generation for Zero-day Polymorphic Worms with Provable Attack Resilience. In *SP '06: Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P'06)*, pages 32–47, Washington, DC, USA, 2006. IEEE Computer Society.

[32] Zhenkai Liang and R. Sekar. Fast and automated generation of attack signatures: a basis for building self-protecting servers. In *CCS '05: Proceedings of the 12th ACM conference on Computer and communications security*, pages 213–222, New York, NY, USA, 2005. ACM Press.

[33] T. Liston. LaBrea: "Sticky" Honeypot and IDS. http://labrea.sf.net, viewed May, 2009.

[34] Andrew Mackie, Jensenne Roculan, Ryan Russell, and Mario Van Velzen. Nimda Worm Analysis. September 2001. ARIS predictor, Attack Registry & Intelligence Service, Incident Analysis Report Version 2, SecurityFocus.

[35] Matthew V. Mahoney. Network traffic anomaly detection based on packet bytes. In *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing*, pages 346–350, New York, NY, USA, 2003. ACM Press.

[36] E. Messmer. Three IPS products pass security evaluation tests. Available at http://www.networkworld.com/news/2006/062606-ips-tests.html. Viewed May, 2009.

[37] David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, and Nicholas Weaver. Inside the Slammer Worm. *IEEE Security and Privacy*, 1(4):33–39, 2003.

[38] David Moore, Colleen Shannon, and Jeffery Brown. Code-Red: a case study on the spread and victims of an Internet worm. In *Proceedings of the Internet Measurement Workshop (IMW)*, pages 273–284, 2002.

[39] David Moore, Colleen Shannon, Geoffrey M. Voelker, and Stefan Savage. Internet Quarantine: Requirements for Containing Self-Propagating Code. In *Proceedings of the 22th IEEE International Conference on Computer Communications (INFOCOM 2003).*, pages 1901–1910, 2003.

[40] J. Newsome, D. Brumley, and D. Song. An End-to-End Self-healing System for Defending against Zero-day Worm Attacks on Commodity Software. Technical Report CMU-CS-05-191, Carnegie Mellon University, February 2006.

[41] J. Newsome and D. Dong. Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software. In *The 12th Annual Network and Distributed System Security Symposium*, 2 2005.

[42] James Newsome, David Brumley, and Dawn Xiaodong Song. Vulnerability-Specific Execution Filtering for Exploit Prevention on Commodity Software. In *Network and Distributed Security Symposium (NDSS)*, 2006.

[43] James Newsome, Brad Karp, and Dawn Song. Polygraph: Automatically Generating Signatures for Polymorphic Worms. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 226–241, Washington, DC, USA, 2005. IEEE Computer Society.

[44] European Network of Affined Honeypots. D0.1: Survey on the state-of-the-Art. http://www.fp6-noah.org/, 2005.

[45] Samuel Patton, William Yurcik, and David Doss. An Achilles' Heel in Signature-Based IDS: Squealing False Positives in SNORT. In *Proceedings of RAID*, 2001.

[46] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(23–24):2435–2463, 1999.

[47] Roberto Perdisci, David Dagon, Wenke Lee, Prahlad Fogla, and Monirul Sharif. Misleading Worm Signature Generators Using Deliberate Noise Injection. In *SP '06: Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P'06)*, pages 17–31, Washington, DC, USA, 2006. IEEE Computer Society.

[48] Phillip Porras, Hassen Saidi, and Vinod Yegneswaran. An Analysis of Conficker's Logic and Rendezvous Points. Technical report, SRI International, March 2009. http://mtc.sri.com/Conficker/, viewed May, 2009.

[49] Georgios Portokalidis and Herbert Bos. SweetBait: Zero-hour worm detection and containment using low- and high-interaction honeypots. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 51(5):1256–1274, 2007.

[50] Georgios Portokalidis, Asia Slowinska, and Herbert Bos. Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation. In *Proceedings of the ACM SIGOPS/Eurosys European Conference on Computer Systems*, pages 15–27, Leuven, Belgium, April 2006.

[51] Moheeb Abu Rajab, Fabian Monrose, and Andreas Terzis. Fast and Evasive Attacks: Highlighting the Challenges Ahead. In *Recent Advances in Intrusion Detection, 9th International Symposium, RAID 2006, Hamburg, Germany*, pages 206–225, 2006.

[52] Sanjay Rawat, V.P. Gulati, Arun K. Pujari, and V. Rao Vemuri. Intrusion Detection Using Text Processing Techniques with a Binary-Weighted Cosine Metric. *Journal of Information Assurance and Security*, 1(1):43–50, 2006.

[53] Benjamin I.P. Rubinstein, Blaine Nelson, Ling Huang, Anthony D. Joseph, Shing hon Lau, Nina Taft, and Doug Tygar. Compromising PCA-based Anomaly Detectors for Network-Wide Traffic. Technical Report UCB/EECS-2008-73, UC Berkeley, May 2008.

[54] Sumeet Singh, Cristian Estan, George Varghese, and Stefan Savage. Automated Worm Fingerprinting. In *6th Symposium on Operating Systems Design and Implementation (OSDI '04)*, pages 45–60, 2004.

[55] A. Slowinska, G. Portokalidis, and H. Bos. Prospector: a protocol-specific detector of polymorphic buffer overflows. Technical Report IR-CS-023, Vrije Universiteit Amsterdam, June 2006.

[56] Anil Somayaji and Stephanie Forrest. Automated response using system-call delays. In *Proceedings of the 9th conference on USENIX Security Symposium*, pages 185–198, 2000.

[57] Stuart Staniford, Vern Paxson, and Nicholas Weaver. How to 0wn the Internet in Your Spare Time. In *Proceedings of the 11th USENIX Security Symposium (Security '02)*, pages 149–167, 2002.

[58] Sam Stover, Dave Dittrich, John Hernandez, and Sven Dietrich. Analysis of the Storm and Nugache Trojans: P2P is Here. In *The USENIX Magazine*, volume 32, pages 18–27. December 2007.

[59] Yong Tang and Shigang Chen. Defending against Internet worms: A signature-based approach. In *Proceedings of the 24th IEEE International Conference on Computer Communications (INFOCOM 2005).*, pages 1384–1394, 2005.

[60] Sandeep A. Thorat, Amit K. Khandelwal, Bezawada Bruhadeshwar, and K. Kishore. Anomalous Packet Detection using Partitioned Payload. *Journal of Information Assurance and Security*, 3(3):195–202, 2008.

[61] Thomas Toth and Christopher Krügel. Accurate Buffer Overflow Detection via Abstract Payload Execution. In *Fifth Internation Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 274–291, 2002.

[62] N. Vanderavero, X. Brouckaert, O. Bonaventure, and B. Le Charlier. The HoneyTank: a scalable approach to

collect malicious Internet traffic. *International Journal of Critical Infrastructures*, 4(1):185–205, 2008.

[63] T. Vogt. Simulating and optimizing worm propagation algorithms,, 2003. Available from http://downloads.securityfocus.com/library/ Worm-Propagation.pdf.

[64] Michael Vrable, Justin Ma, Jay Chen, David Moore, Erik Vandekieft, Alex C. Snoeren, Geoffrey M. Voelker, and Stefan Savage. Scalability, fidelity, and containment in the potemkin virtual honeyfarm. *SIGOPS Oper. Syst. Rev.*, 39(5):148–162, 2005.

[65] David Wagner and Paolo Soto. Mimicry attacks on host-based intrusion detection systems. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 255–264, New York, NY, USA, 2002. ACM Press.

[66] Helen J. Wang, Chuanxiong Guo, Daniel R. Simon, and Alf Zugenmaier. Shield: vulnerability-driven network filters for preventing known vulnerability exploits. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 193–204, New York, NY, USA, 2004. ACM Press.

[67] Ke Wang, Gabriela Cretu, and Salvatore J. Stolfo. Anomalous Payload-Based Worm Detection and Signature Generation. In *Recent Advances in Intrusion Detection: 8th International Symposium (RAID)*, pages 227–246, 2005.

[68] Ke Wang, Janak J. Parekh, and Salvatore J. Stolfo. Anagram: A Content Anomaly Detector Resistant to Mimicry Attack. In *Recent Advances in Intrusion Detection, 9th International Symposium (RAID), Hamburg, Germany*, pages 226–248, 2006.

[69] Ke Wang and Salvatore J. Stolfo. Anomalous Payload-Based Network Intrusion Detection. In *Recent Advances in Intrusion Detection: 7th International Symposium, RAID 2004, Sophia Antipolis, France*, pages 203–222, 2004.

[70] XiaoFeng Wang, Zhuowei Li, Jun Xu, Michael K. Reiter, Chongkyung Kil, and Jong Youl Choi. Packet vaccine: black-box exploit detection and signature generation. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 37–46, New York, NY, USA, 2006. ACM Press.

[71] Xinran Wang, Chi-Chun Pan, Peng Liu, and Sencun Zhu. SigFree: a signature-free buffer overflow attack blocker. In *USENIX-SS'06: Proceedings of the 15th conference on USENIX Security Symposium*, pages 225–240, 2006.

[72] M. Williamson. Throttling Viruses: Restricting Propagation to Defeat Malicious Mobile Code. Technical Report HPL2002-172, HP Laboratories Bristol, 2002.

[73] Vinod Yegneswaran, Paul Barford, and David Plonka. On the Design and Use of Internet Sinks for Network Abuse Monitoring. In *Recent Advances in Intrusion Detection: 7th International Symposium, RAID 2004, Sophia Antipolis, France*, pages 146–165, 2004.

[74] Vinod Yegneswaran, Jonathon T. Giffin, Paul Barford, and Somesh Jha. An Architecture for Generating Semantics-Aware Signatures. In *Proceedings of the 14th conference on USENIX Security Symposium*, pages 97–112, 2005.

Author Biographies

Craig Smith completed his M.Eng at Carleton University in Ottawa, Canada. His areas of interest include network security, networked applications, pattern classification and cryptography. **Ashraf Matrawy** received the B.Sc. and M.Sc. degrees in computer science and automatic control from Alexandria University, Egypt, and the Ph.D. degree in electrical engineering from Carleton University, Ottawa, ON, Canada. Ashraf is currently an Assistant Professor at Carleton University. He is a senior member of the IEEE, serves on the editorial board of the IEEE Communications Surveys and Tutorials journal, and has served as a technical program committee member of a number of IEEE and other international conferences. Ashraf's research interests include reliable and secure computer networking, and analysis of Internet traffic.

Stanley Chow works in the Enabling Computing Technology domain of Bell Lab. He received his B.Sc. from University of British Columbia. His work experience include telephony systems, ATM systems, silicon CAD/CAM, IT systems, tamper- resistant software, DRM, network security. His research interests include security in general (including network and software), software engineering, real-time systems, and algorithms. He co-founded a software security company (that has since been acquired). He is currently concentrating on bringing together research and products. He is a Senior Member of IEEE and holds PMP, CISSP and SANS GAWN certifications. **Bassem AbdelAziz** has a Bachelor degree in Electrical Engineering from Mansura U. (1993), and a Master of Applied Science degree from Ottawa U (2003). He is a member IEEE, and a member of the Professional Engineers Ontario (PEO). Bassem has over fifteen years of experience in software engineering, computer networks, signal processing, and network security. He holds several granted and pending patents, and has published papers on multimedia security, malware detection, and network security. Bassem is currently a senior software engineer at Trillys Systems.