Fighting Fast Worm Epidemics: Prevention, Detection, and Neutralization

고려대학교 김효곤

1. Introduction

Today's worm epidemics entered a regime where their spread is so overwhelmingly fast that traditional human-intervened response is no longer adequate. For instance, the SQL Slammer (a.k.a. Sapphire) epidemic infected most of the vulnerable nodes in just 10 minutes [1], while substantial response came in 2 to 3 hours world-wide [2]. Thus the response only stopped side-effects, not the worm itself. From the epidemiological model [3], we can readily prove that the infection speed is a function of, most importantly, the scanning rate and the vulnerable population size. Namely, the logistic equation below dictates the dynamics of the epidemic:

$$x(t) = \frac{K}{1 + \left(\frac{K}{x_0} - 1\right) e^{-rt}} \qquad (1)$$

where x(t) is the number of infected nodes at time t, K is the total number of vulnerable nodes, and x(t=0)=x0 is the number of worms at the outset. r=K/T*v is the infection rate, where v is the scanning rate, and T is the scanned space size. Since worms typically scan the entire IP address space, T=232. Figure 1 shows the time to 90% infection as a function of K, given v=10,000/s (c.f. in SQL Slammer, the average was 4,000/s and the recorded maximum was 26,000/s [1]), and x0=1 (i.e., worst condition for the worm).

In essence, for a fast-scanning worm exploiting

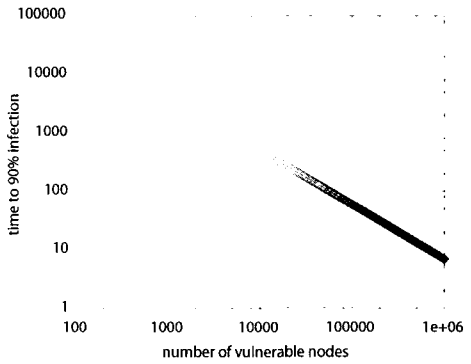

Fig 1 Speed of epidemic for v=10,000

any vulnerable software with a substantial installed base (e.g. >100,000), the peak epidemic is attainable in just a few tens of seconds. Currently, against such fast worms we have absolutely no impeding element installed in the Internet infrastructure. Although there is a proposal for a signature-based content filtering inside network [4], its practicability is dubious due to the complexity and its inherent limitation against encryption and polymorphism that is increasingly employed by modern wormsand viruses (e.g. Loveletter [5]; There are even worm generator softwareson the Web that automatize polymorphic worm generation [6]).

In this paper, we explore three methodologies that respectively addresses the prevention, detection, and countering of fast worm epidemics. Section II analyzes the effectiveness of destination IP address filtering against fast worm epidemics. Section III presents a novel approach to detect the

behavioral characteristics of worm epidemic, victim scanning. Section IV explores the idea of worm-killing worm and investigates its practicability against fast worm epidemics. Seciton V concludes the paper.

2. Destination address filtering

Almost without exception, worms randomly scan for victims [1,7] (although so called "hit-list" worm [3] accumulates the list of vulnerable nodes before the outbreak, in no major epidemic so far the captured worm body contained one). For instance, Code Red II generates an address within the same /8 prefix with probability of 1/2, /16 prefix with 3/8, others with 1/8 [7]. SQL Slammer does not have such preference, and it generates a random address with equal probability. The randomly generated IP address of a possible victim in this way can be illegal in one of two ways Martian [8] or currently unallocated (a.k.a. "tarpit") [9]. Collectively, these illegal destination addresses occupy 44% of the entire IP address space [10]. (Although these blocks are subject to future allocation, the procedure is slow enough [11] to reflect on the address checker.) Therefore, more than 2 out of 5 infection attempts through random scanning result in the violation of the address usage, and we can leverage this property to detect and filter them on the packet level. Furthermore, a host can be made to "quarantine" itself if it transmits a Martian or unallocated destination address more than a preset number of times, say , in a given observation interval. In the quarantine, the host launches a self-audit, eventually killing the worm within. We can set fairly low (e.g. 5) since in reality, there are few innocent hosts that habitually transmit to illegal addresses [10]. We will assume that the destination address filtering includes the quarantine as well as the packet-level filtering.

Suppose d is the fraction of vulnerable nodes that employ the destination based filtering. The address filtering effectively reduces the vulnerable population base by a factor of (1-d). Thus the infection rate in Eq. (1) changes to:

$$r' \approx (1-d)r \qquad (2)$$

.The approximation is because we allow scans before we quarantine the perpetrator. Note that K is the number of networked servers on the Internet (whose open service a malicious code can exploit). Today, only Web servers and possibly some P2P "servers" could be on the order of millions. Even then, K/T should be much less than 0.001. With such small K/T, the probability of an infection in scans is: $1-\left(1-\frac{K}{T}\right)^{\theta} \approx \frac{K\theta}{T} \approx 0$, so we ignore it in Eq. (2). On the other hand, the probability of a perpetrating node not caught in random scans is approximately [10]: $p_n(N,\theta) = \sum_{i=0}^{\theta-1} \binom{N}{i} 2^{-N}$ which is a fast decreasing function of N [10]. N=vL is the number of scan packets in a given observation interval L. Since v is so high in fast epidemics, enough to cause denial-of-service [1], the "impunity" probability above is negligible. Now we show how much time this mechanism buys us at the outset in which to react to an epidemic. Figure 2 plots the time to 10% infection as a function of d for four different values of K, ranging from 1,000 to 1,000,000. We notice that the address filtering effect becomes visible only at high level of d. So it will not help much to slow down a fast epidemic

Fig 2 Timescale of initial outbreak as a function of d

with partial deployment. However, as d approaches 1, it can arbitrarily impedethe epidemic even for large K. In particular, for d=1, x(t) in Eq. (1) (subject to the modification of Eq. (2)) remains at 1, independent of t.

So the whole issue is reduced down to whether sweeping deployment is feasible or not. We argue that it is feasible. Whenthe operating system platforms in the past large epidemics are limited to a small number and only one is exploited at a time, its deployment at hosts can be as sweeping and swift as in everyday operating system patch. For instance, Microsoft Windows operating systems regularly perform such automaticupdate, and some Linux publishers also provide automatic patches on demand [12]. And the update can also deal with the changes of illegal address blocks due to future allocation [11]. Finally, as for source address filtering [13] and the filtering at IPv4 routers [8], a standard or a "Best Current Practice (BCP)" could also be established for the destination address filtering at hosts.

3. Fast Classification, Calibration, and Visualization of Network Attacks

Detecting attacks on backbone-speed links, let alone performing attack classification and other more involved tasks, is hard. The formidable speed forbids any algorithm requiring more than a few memory lookups and computation steps per packet, to operate in-line. The sheer volume of the traffic would dwarf even the fiercest attacks, rendering it hard to separate them out. Therefore, any algorithm for backbone-speed links must simultaneously achieve speed and sensitivity, which, unfortunately, are usually in a trade-off relation. But if only feasible, attack monitoring on a backbone link can lead to unique advantages, first for detection and then for possible preemption. For instance, a global-scale hostscan activity is far more defined and visible in the backbone before it spreads out towards targets. Likewise, distributed denial-of-service (DoS) flows first converge in the backbone, rendering themselves more conspicuous, before they proceedand bombard the victim.

On each packet arrival, we want to judge whether it is (highly likely) part of an attack or not. And if indeed it constitutes an attack, we want to classify the type of attack: DoS, hostscan, or portscan. Furthermore, we want to identify who is the victim (DoS), who is the perpetrator and what ports are scanned (hostscan, portscan), and the intensity of the attack. In this section, we discuss our approach to achieve these goals.

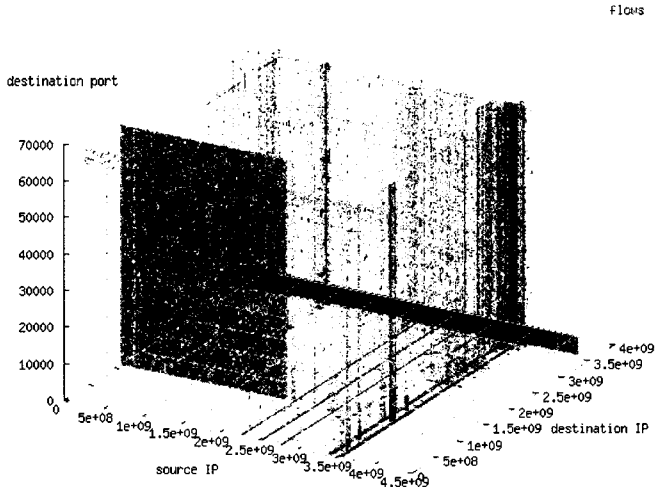

Figure 3 Flows at around 9:35 a.m., Dec. 14th, 2001.

First, we define a flow to be a 3-tuple <s, d, p>, composed of the source address (s), destination address (d), and destination port (p). Our novel idea starts from the observation that only DoS attack, hostscan and portscan appear as a regular geometric entity in the hyperspace defined by the 3-tuple. For instance, source-spoofed DoS packets maintain a fixed destination address, thus appears as a straight line (in case destination port is fixed) parallel to the s axis, or as a rectangle (in case destination port is randomly varied) parallel to the s-p plane.Legitimate flows, on the other hand, appear as random points scattered across the hyperspace.

Figure 3 shows the flows observed at 9:35 and 9:36 a.m. in December 14th, 2001 on two trans-pacific T-3 links connecting the U.S. and a Korean Internet Exchange. The three axes are the source IP address, destination IP address, and destination port as used in the flow definition above. (The source and the destination addresses have decimal scale.) Each dot in the 3-dimensional hyperspace represents a single flow (not a packet). Total of 2.22 million packets were mapped to the hyperspace in the figure, where the packets in the same flow fall on the same position. We can easily recognize the regular geometric formations, such as a large rectangle and a leaner rectangle lying parallel to s-axis, lines parallel to d-axis, and numerous vertical lines. These regular formations are (destination port varied) DoS attacks, hostscans, and portscans, respectively. Although far outnumbering them, legitimate flows do not form any regular shape, and are less conspicuous.

By focusing on only the regular formations, we can weed out the majority legitimate traffic, drastically reducing the amount of data we need to process. This not only helps the detection algorithm keep up with the traffic speed, but also boosts the accuracy. This is roughly how we break the aforementioned trade-off between speed and sensitivity. As one might notice, the identification of regular patterns in the hyperspace lies in the center of the approach. Instead of employing complex pattern recognition techniques such as 3-dimensional edge detection, we apply an original algorithm that captures the "pivoted movement" in one or more ofthe 3 coordinates. This is because, from graphical perspective, such movement forms the aforementioned regular pattern along the axis of the pivoted dimension. In hostscan, the source IP address and the destination port are fixed, while the destination IP address pivots on them [1]. In portscan, the destination port pivots on the source and the destination IP address. In source-spoofed DoS, the destination IP address is fixed, while either only the source IP address or both the source IP address and the destination port pivots on it [14].

In order to detect the presence of pivoting in the traffic stream, our scheme first generates a signature for each incoming packet. The signature is simply a tuple consisting of 3 binary values: <*Ks, Kd, Kp*>. The coordinates in the signature one-to-one correspond to the flow coordinates. Each coordinate value in the signature tells us whether the corresponding value in the flow (that the packet in hand belongs to) was seen "recently" or not. (The degree of recentness for different coordinates could vary, and we will deal with it later.) For example, suppose two flows

Arrival time	Flow	Flow ID
t :	<3.4.5.6, 5.6.7.8, 90>	1
t+1 :	<1.2.3.4, 5.6.7.8, 80>	2

pass through the monitorthat executes our scheme. For convenience, throughout the paper we will call the monitor RADAR monitor (for Real-time Attack Detection And Report), and the algorithm that it executes, RADAR algorithm. Unless we explicitly mention the algorithm, we refer to the monitor (that includes the algorithm) when we simply say RADAR. RADAR remembers these two flows for a finite time duration L. For the sake of

explanation, let us assume for now that the time duration is the same for every coordinate, e.g., L = 2. When a packet with source IP = 1.2.3.4, destination IP = 3.4.5.6, destination port = 90 appears at time t+2, RADAR tells that this packet's signature is <Ks, Kd, Kp>=<1, 0, 1>. This is because source IP address 1.2.3.4 appeared in flow (2) and port 90, in flow (1). But 3.4.5.6 was not used either in (1) or (2) as the destination address, so Kd = '0'. If L = 1, flow (1) would have been purged from RADAR at the time of the packet arrival, and the signature would be <1, 0, 0>.

In principle, this per-packet signature determines whether the packet is part of a "pivoted movement", and if so, what type it is. Note that when pivoting occurs, the value of the pivoted coordinate changes constantly from packet to packet within the attack stream. From the perspective of RADAR algorithm, the pivotedcoordinate is viewed as persistently presenting recently unobserved values. So RADAR will keep generating <1, 0, 1> signatatures for hostscan. This way, RADAR gets to yield the signatures <1, 0, 1>, <1, 1, 0>, or <0, 1, *> rather frequently in the presence of hostscan, portscan, or source-spoofed DoS, respectively. ('*' is wildcard, i.e., '0' or '1'). These signatures are what we call attack signatures, and the corresponding flow goes through further examination. Sometimes legitimate traffic can get attack signatures, and vice versa. Or one attack might be mistaken as another, all due to hapless modification of one or more coordinates in the signature, so some refinement is required in back-end processing (which is much less time-pressed). The accuracy of the proposed algorithm thus depends on how likely these unwanted changes in the signature are, so this statistical aspect of our algorithm is analyzed in [15]. Note that each new value in the pivotedcoordinate means a new flow in our flow definition. Once the number of newly seen flows exceeds a preset threshold per source (perpetrator) or destination (victim) depending on the type of attack analyzed in the back-end, alarm goes off so that any necessary action can be taken.

Table I exhaustively enumerates all signatures and their conceivable implied attack types. As we described earlier, '0' in a signature means that the monitor has not recently seen the value in the given coordinate. Thus, if a packet belongs to an attack stream, '0' value in a coordinate most probably means that the coordinate is pivoting. The leftmost column is the number of dimensions that are pivoting. The second column is how the attacks might manifest themselves geometrically when the attack is mapped on to the 3-d hyperspace a la Figure 3. An important note here is that the signatures listed in Table I are self-induced. Namely, the values in a signature are what are caused by the corresponding attack itself, but not by others. To wit, these are what an attack would obtain in the absence of any cross (legitimate + other type of attack) traffic. But as we discussed earlier, cross traffic might overlap in one or more coordinates, and these signatures are not always those detected when correponding attack is under way. For <0, 0, 0>, one or more coordinates can be flipped to 1 by cross traffic that happens to coincide on IP addresses or port number. Suppose a flow <4.4.4.4, 2.2.2.2, 5555> is initiated after a flow <1.1.1.1, 2.2.2.2, 3333> is registered by RADAR. Then the former will receive <0, 1, 0> signature, which RADAR recognizes as the port-varied DoS attack.

With careful application of these signatures, we can find the attacks in the traffic with low false positive probability and high sensitivity [15]. Figure 4 shows an example of this graphical representation of the attacks, as extracted by RADAR algorithm from the trace data of Figure 3.

The memory requirement of the hash tables in the main filter and the post filter is moderate. Assuming we use a 24-bit hash for the source and destination IP tables, we need at least 225 hash buckets whose heads are a pointer (usually 4 octets). This alone is 128MB. Over and above, we

need to store each flow in these tables, where a flow has at least 2 IP addresses, 1 port number, and a timestamp. Also each entry needs a pointer to the next entry. So each flow entry requires at least 17B. Assuming there are 1 million flows being tracked simultaneously, 34MB should be used. If memory is a critical resource, we could use 23-bit hash, halving the requirement, and then 22-bit hash and so forth.

4. On the functional validity of the worm-killing worm

At the extreme of the spectrum of automatic responses is the elusive killer-worm. The notion of worm-killing worm has been in the folklore for some time [16,17,18]. Although there is no formal definition of killer worm, its most characteristic (and controversial) aspect is that it spawns exactly as worms do. In fact, it isa worm, except that it cures the infected and preventively patches vul-

TABLE 1 Attack signatures

Dim.	Graphical manifestation	Signature	Implied attack
0	Dot	<1, 1, 1>	Single-source-spoofed DoS
1	Straight line	<1, 1, 0>	Portscan
		<1, 0, 1>	Hostscan
		<0, 1, 1>	Source-spoofed DoS (destination port fixed)
2	Rectangle	<1, 0, 0>	Kamikaze
		<0, 1, 0>	Source-spoofed DoS (destination port varied)
		<0, 0, 1>	Distributed hostscan
3	Hexahedron	<0, 0, 0>	Network-directed DoS

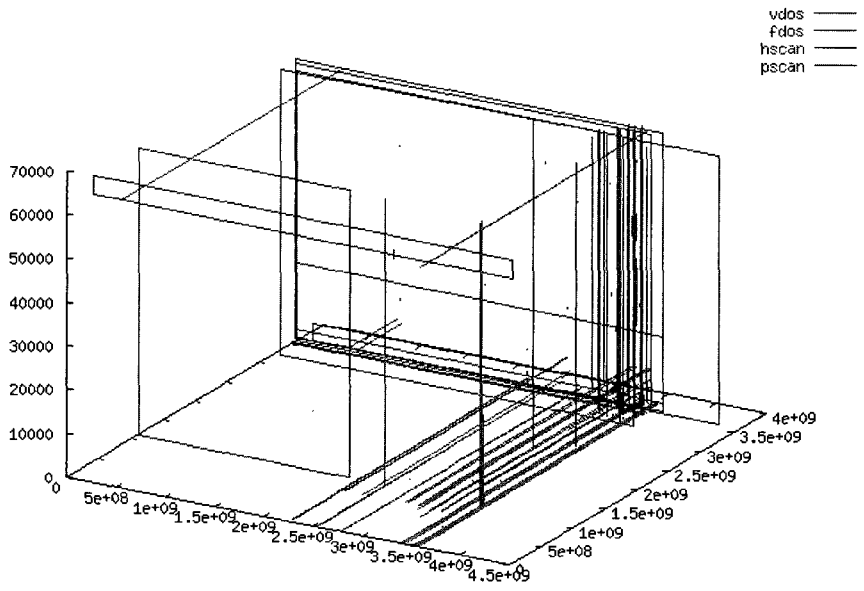

Figure 4 A RADAR-processed result of Figure 1

nerable nodes. The gist of the idea is that through and only through worm-like spawning, it can par with fast worms Henceforth we will refer to the worm-killing worm as "the killer worm" or "the killer,"as opposed to its malicious counterpart, which we refer to as simply "the worm." in speed. However, the very idea of unleashing a self-propagating code possibly over administrative boundaries, however "good-willed" [16] it might be, can be threatening. What if the killer worm itself is compromised? How complicated is it to set up the trust association among normally mutually distrustful administrative domains?

Despite these concerns, there was a report [17] that two separate killer-worm mobile codes, dubbed Code Green [18] and CRclean, were actually released to the Internet to fight the Code Red (obviously without consent from the "victims"). And recently, Toyoizumi and Kara [16] attempted a theoretical modeling of the killer worm based on the Lotka-Volterra equations in order to optimize its behavior. In this paper, too, the ethics and other non-technical ramifications are put aside. This is because we believe that judging its functional validity should come first before starting any such discussions, especially in the light of recent fast worm epidemics such as that of SQL Slammer. So in this paper we attempt to quantify the following aspects of the killer worm:

- effectiveness (i.e., if a killer worm can indeed preempt a worm epidemic, and if so, how fast)
- efficiency (i.e., at what costs)

In our system model, there are a finite number of susceptible nodes. In reality, these are computers with an exploitable vulnerability. For instance, they could be Web servers running Microsoft IIS software (as in Code Red II) or SQL server software (as in SQL Slammer). Usually, the number of susceptible nodes is much smaller than the size of the entire population, which is the number of hosts attached to the Internet in reality. When a susceptible node is compromised by the worm released by the attacker, it becomes infected. On the other hand, when a previously infected node is cured, we say it is removed of the worm. This model closely resembles that used in epidemiology, and we can use a differential equationto mathematically describe the dynamics. In Staniford et al. [3], a logistic equation is used to model the number of infected nodes after a given time since the outbreak began. When the killer worm is injected into the system, the dynamics becomes more complex. In [16], the Lotka-Volterra equations are employed to model the prey-predator dynamics between the worm and the killer worm. Unlike in [3], however, the environmental capacity, i.e., the number of susceptible population that the worm can prey upon, is not included in the model.

When the system starts, a single worm begins to spawn. It scans the entire population (since it does not know a prioriwhich node is vulnerable) to find a susceptible node An exception is the "hit-list"worm, which accumulates the list of susceptibles through scanning before it starts spreading [3]. So far, in no major worm epidemic the worm body contained a hit list (For instance, see [1]). When the worm finds one, it infects it, which involves self-replicating and planting a replica there. The replica at the infected node also starts spreading. The dynamics of the epidemic is affected by many parameters: the total population (T), the number of initial susceptible population (S), the scanning rate (v), and the time for the infected node to become active (d). In particular, if d=0, the infection rate of a worm under random scanning is

$$i = v \cdot \frac{S}{T} \tag{3}$$

When the killer worm is used, we must also consider the reaction time (a) and the killer's scanning rate (k). The reaction time is what takes to detect the onset of the epidemic and take a counter-action (e.g., unleashing the killer worm). It is well known that reducing the reaction time is

TABLE 2 Default parameter values

Parameter	Meaning	Default value
T	Total population	232
S	Susceptible population	100,000
v, k	Scanning rate	10,000/s
d	Worm activation delay	0s
a	Reaction time	10s
W	Number of starters	1

critical [4] to contain the epidemic to the minimum. In this paper, we assume that once the killer is unleashed, it replicates itself on both susceptible and infected nodes. Other mode of operation may be replicating only on the previously infected nodes.

The shortcoming of the prey-predator model of [16] is that it does not explicitly take account of these parameters, above and beyond the environmental capacity. Therefore, in this paper we mostly resort to the simulation experiments to investigate their affects.

We will use the following values for the system parameters:

As for T, note that it does not have to represent the real population in the system. Real-life worms usually probe the entire IP address space in a blind manner [1,7], not knowing which address is in real use and vulnerable. Therefore, it is natural to model the worm as well as the killer worm to "think" that total population is 232 (entire IP space size). The number of susceptible nodes S is set roughly to the order of SQL Slammer infection which was at least 75,000 [1]. Also for v and k, the SQL Slammer epidemic is used as a reference. In SQL Slammer, the average scanning rate from the infected node was 4,000/s where the observed maximum was 26,000/s [2]. Since our focus in this paper is on the effectiveness of the killer worm in fast epidemics, we set the average scanning rates higher. We do not set k > v since in a fast epidemic, the worm scanning rate is likely to be at the maximum that the infected node's processing and bandwidth capacity can offer [1]. So the killer worm can at best par with it, but not exceed. As for d and a, we set the default to 0 and 10 seconds, respectively. But we will see the impact of changing these values below. By default we assume that both the worm and the killer startfrom a single entity as mentioned in the previous section. So the number of progenitors W is 1 unless otherwise mentioned. In reality, however, this may not be true. Both the attacker and the killer worm defense system may each want to start from as many locations to boost the initial spawning speed. We will discuss this aspect later on.

In the first experiments, we investigate the impact of the parameters to the dynamics of the epidemic, starting from the reaction time, a. Just as in any infectious disease control, containing the outbreak in its early stage is considered crucial [4]. For instance, in retrospect, it is argued that we had 30- to 60-second window atthe beginning of the SQL Slammer epidemic for effective containment [2]. In this paper we are looking at faster epidemics, so we vary a well within that range: from 5 to 30 seconds. Figure 5 shows the result. The x-axis represents a, while the y-axis is the number of infected nodes at the peak of the epidemic, Imax. We find that Imax is roughly a linear function of a. In terms of the prevention

effect, the number of nodes patched by the killer before infection nearly disappears with a > 30. Namely, almost all vulnerable nodes in the system are infected before the killer visitsin that time. It corroborates the analysis on the size of the time window for effective response in a fast epidemic [2]. This result has a grave implication in future fast epidemics, detection must be done extremely fast, in much less than a minute. Otherwise, most vulnerable nodes will have already sustained a possibly damaging hit from the worm by the time a killer worm comes to the aid.

But how easy is it to detect the onset of the outbreak within, say 10 seconds, without too many false positives, considering there are only a small number of infected nodes in that time ? Designing a detector with such precision and speed will be extremely difficult, if not impossible. Unless set loose within a few tens of seconds into the epidemic, even the powerful killer worm cannot prevent massive infection. Therefore, unleashing the killer worm after detecting the outbreak is likely to have limited utility against a fast worm. But we reserve the judgment on its effectiveness on slow epidemics.

In terms of the bandwidth usage, a killer worm that keeps trying to spawn even after the epidemic dies out will be no less problematic than the worm epidemic itself. Recollect that SQL Slammer worm could paralyze a substantial part of the Internet with just scanning traffic (it carried no other damaging payload). In [16], this very issue is addressed and a solution is proposed controlling "predatory rate" and "predator multiplication rate". The former is the rate at which a killer worm hunts (thus kills) a malicious worm, and the latter is the rate at which a killer replicates itself upon a kill. For instance, in our system the multiplication rate is 1, i.e., a killer worm that found an infected node gives birth to a single replica. Unfortunately, in reality we cannot determine the predatory rate a priori it can only be obtained through a post mortem analysis, e.g., 8.5 per second in SQL Slammer [7]. This is because the predatory rate is a function of many system parameters we do not know before the outbreak. For one, we do not know S, namely how many have not patched for the particular vulnerability, before the epidemic actually occurs.

Therefore, we need a more practical mechanism to optimize the traffic usage of the killer worm. In this paper, we fuse the "rumor-mongering" model [19] into the construction of the killer worm: a rumor-monger loses incentive to spread the word when he finds that too many people already heard about it. This way, by the time most people have heard the rumor, the rumor naturally stops spreading. Likewise, if the modified killer finds that

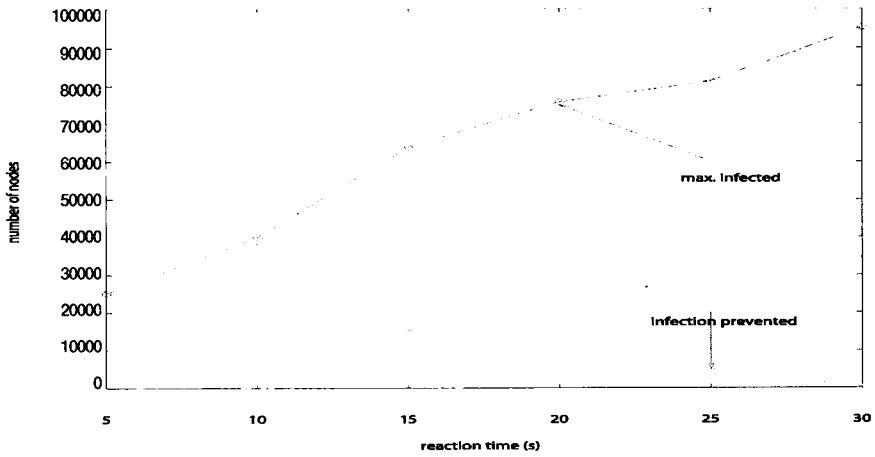

Figure 5 The impact of the reaction time

the visited node has been removed of the worm already, it considers the visit a failure. When the fraction of the failures increases beyond a threshold r, the killer (to be precise, an instance) self-destroys. Figure 4 shows the impact of this modification on the system behavior. The y-axis represents the number of killers' scanning packets generated system-wide. We vary r from 0 to 0.9, and the impact is significant. Even at r = 0.1, the number of scanning packets decreases by half. However, higher thresholds cause less dramatic decrease. But it is one required self-control mechanism that a killer worm must be equipped with.

Would the infection be more virulent with more stringent killer control (i.e., larger r)? Surprisingly, the number of infected nodes for different values of r is almost identical in our current setting [20]. Although not shown due to space constraint, our investigation [20] shows that the time the curves in Figure 6 begin to diverge interestingly concurs with the time at the peak infection. Although Figure 6 seems to suggest that lowering r is only beneficial, the optimal value of r must be a function of many other system parameters. Again, we need a general analytical framework to dynamically determine the optimal value of r. It is a subject of our ongoing work.

In order to minimize the epidemic, the killer worm must hit as many infected nodes as possible at the outset, when the number of the infected node is still small. One way to achieve this objective is to improve the accuracy of the "guesses"on the part of the killer. To assist the killer, we can let each host maintain a list of recent correspondents. For instance, a 30-second worth of IP addresses from which a host received packets (possibly containing worm payload) could be recorded. When the killer worm finds an infected node, it can start from the IP addresses in the node's list since the worm that planted a replica at the infected node could have come from one of those addresses. Figure 7 shows the result of this enhancement. We notice that the infection is indeed significantly reduced, while the reduction quickly becomes

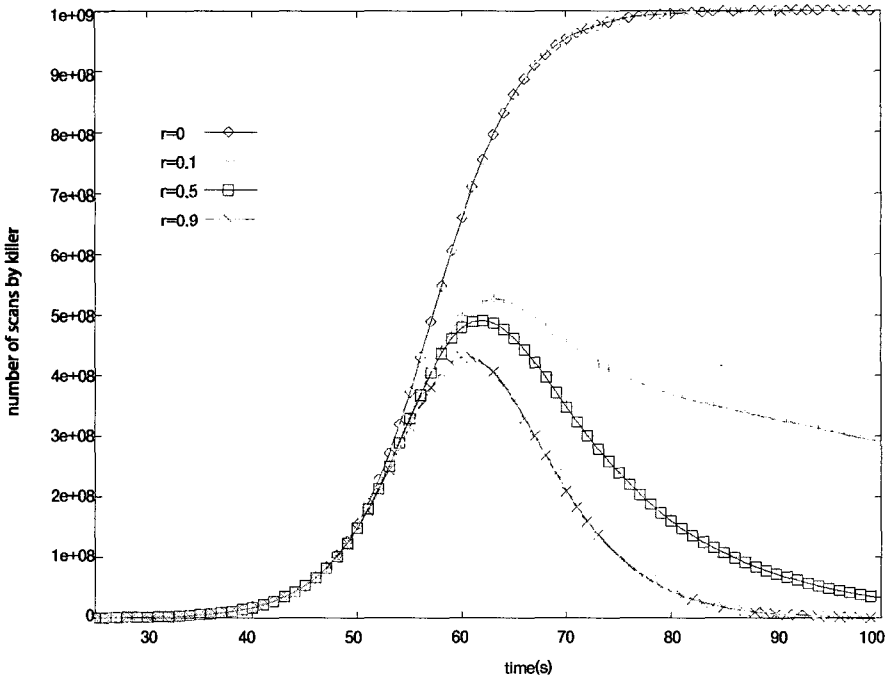

Figure 6 Volume of scanning traffic with different r

marginal with progressively large time window.

One might think that the notion of "time bomb" is applicable on the attacker's part namely, if the attacker intentionally puts a delay until the activation of the worm, e.g., 60 seconds, then the worm would evade the backtracking. However, this delayed activation technique would only procrastinate the spread of the worm in its initial phase [20]. In the figure, the delayed epidemic still shows the exponential behavior (the y-axis is in log scale), but it is too slow to cause a fast epidemic. This is because the attacker, by inserting the artificial delay, is widening the time gap between the "generations" [3] of infection. So we can argue that fast epidemics will shun such delay. If deployed widely, therefore, exploiting the history of communication in killer worm propagation will be a powerful technique to boost the killer's performance.One last caveat is that this technique is effective against the worms that are carried over TCP, and those that use UDP but do not employ source address spoofing. This is because a bogus address in the list does not help in the backtracking. Note that the scanner normally cannot spoof when using TCP.

5. Conclusion

In this paper, we explore methodologies to prevent, detect and fight fast worm epidemics. Destination IP address filtering exploits the fact that most current worms generate illegal destination addresses with a high probability. We find that its effectiveness is guaranteed only under wide deployment. We argue that it is not as impractical as one might think, if we can utilize the OS patch channels. The update will be simple, swift, and sweeping, and it will fundamentally block so called fast epidemics. However, all bets are off if worms follow suit, namely, if worms censor the randomly generated destination address. Second, we consider how we can capture the characteristic worm

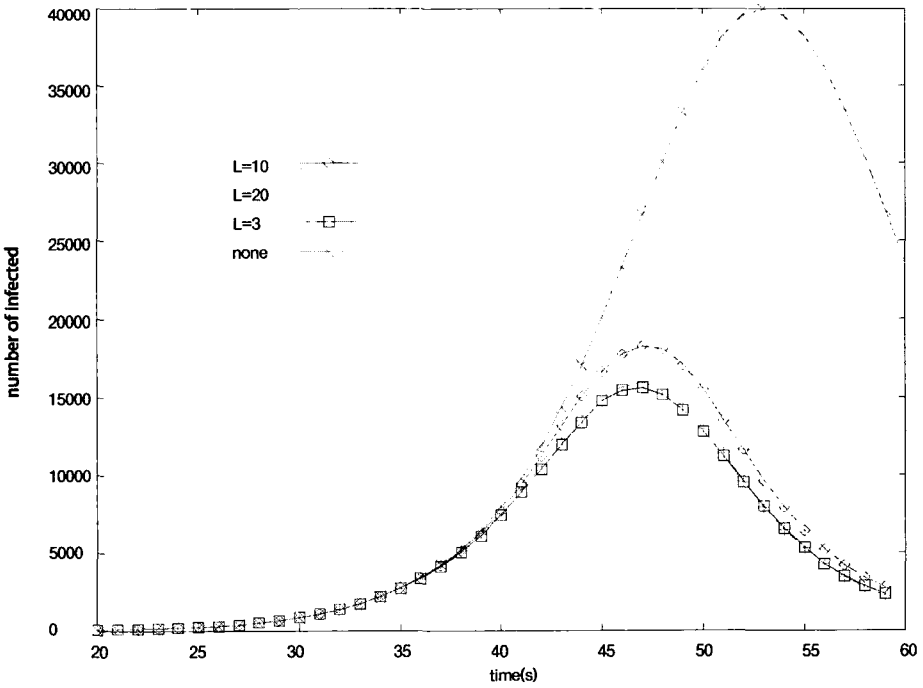

Figure 7 The impact of address recording for backtracking

scanning behavior on packet level at high-speed. A novel algorithm called RADAR (real-time attack detection and reporting) is shown to have low false positive probability and to operate with minimal memory access overhead. Although precise and sensitive, it requires further refinement to distinguish the early sign of worm epidemic and casual hostscan. Third, we explore the practicability of the elusive notion of worm-killing- worm. Although powerful, it is shown to be useful only when it is launched early in the epidemic, which is infeasible due to the difficulty of early detection and patch generation. Considering other undesirable aspects such as excessive traffic caused by the killer worm itself, it is considered not a viable option against fast worm epidemics. In summary, there is no solution or solutions that solves the problem of fast worm epidemics. Further research for more effective countermeasures is badly needed.

Acknowledgements: This work was supported by Korea University Grant.

References

[1] CAIDA, "Analysis of the Sapphire Worm," http://www.caida.org/analysis/security/sapphire/, Jan. 2003.

[2] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "The spread of the Sapphire/Slammer worm," a NANOG presentation, http://www.nanog.org/mtg-0302/ppt/worm.pdf, March, 2002.

[3] Stuart Staniford, Vern Paxson, and Nicholas Weaver, "How to Own the Internet in Your Spare Time," 11th USENIX Security Symposium, June, 2002.

[4] D. Moore, C. Shannon, G. Voelker, and S. Savage, "Internet Quarantine: Requirements for Containing Self-Propagating Code", IEEE Infocom, March, 2003.

[5] S. Taylor, "The Complexities of Viral VB Scripts," European Institute for Computer Antivirus Research (EICAR) International Conference, 2001.

[6] VBS Worm Generator, http://vx.netlux.org/vx.php?id=tv07.

[7] CAIDA, "CAIDA analysis of Code Red," http://www.caida.org/analysis/security/code-red/.

[8] F. Baker, Requirements for IPv4 routers, RFC 1812.

[9] IANA, Internet Protocol Version 4 Address Space, http://www.iana.org/assignments/ipv4-address-space.

[10] H. Kim and I. Kang, "On the Effectiveness of Martian Address Filtering and its Extensions," IEEE Globecom, Dec. 2003.

[11] A. Broido, E. Nemeth and K. C. Claffy, "Internet Expansion, Refinement, and Churn," a NANOG presentation, Feb. 2002.

[12] Red Hat Linux, http:// http://sources.redhat.com/.

[13] P. Ferguson and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing," RFC 2827.

[14] K. Houle and J. Weaver, "Trends in Denial of Service Attack Technology," CERT Coordination Center, Oct. 2001.

[15] H. Kim, "Fast Classification, Calibration, and Visualization of DoS and Scan Attacks for Backbone Links," Technical Report, June 2003, http://net.korea.ac.kr/papers/RADAR.html.

[16] H. Toyoizumi and A. Kara, "Predators: good will mobile codes combat against computer viruses," New Security Paradigms Workshop 2002, Sept. 23-26, Virginia Beach, USA.

[17] Newsbyte, http://www.newsbytes.com/news/01/169707.html

[18] "D. HexXer," Code Green, http://www.securityfocus.com/archive/82/211428.

[19] A. Demers et al., "Epidemic algorithms for replicated data management," Sixth Symp. on Principles of Distributed Computing (Van-

couver), ACM, August, 1987, pp. 1-12.
[20] H. Kim, "Demystifying the killer worm," Techreport, Korea University, Aug. 2003. http://net.korea.ac.kr/killer.html.

김 효 곤

1987. 2 서울대학교 공과대학 전자계산기공학과(학사)
1989. 2 서울대학교 대학원 컴퓨터공학과(석사)
1995. 12 미국 펜실베니아 대학교 컴퓨터 및 정보과학(박사)
1996. 2~1999. 9 미국 벨 통신 연구소 인터넷 구조 연구실 연구원
1999. 9~2003. 1 아주대학교 정보통신대학 정보 및 컴퓨터공학부 조교수
2003. 3~현재 고려대학교 정보통신대학 컴퓨터학과 부교수
관심분야 : 인터넷, 네트워 보안, 이동통신
E-mail : hyogon@korea.ac.kr