Real-Time Visualization of Network Attacks on High-Speed Links

Hyogon Kim, Korea University, Inhye Kang, University of Seoul, Saewoong Bahk, Seoul National University

Abstract

This article shows that malicious traffic flows such as denial-of-service attacks and various scanning activities can be visualized in an intuitive manner. A simple but novel idea of plotting a packet using its source IP address, destination IP address, and the destination port in a 3-dimensional space graphically reveals ongoing attacks. Leveraging this property, combined with the fact that only three header fields per each packet need to be examined, a fast attack detection and classification algorithm can be devised.

Intuitive visualization of ongoing attacks is something that network administrators can make good use of, providing quick perceptual clues before other more complicated analyses kick in. We demonstrate in this article that such attack visualization can be surprisingly straightforward yet highly informative. Although there are numerous types of malicious attacks against or via the network, denial-of-service (DoS) attacks and worm epidemics undoubtedly have been the most notorious recently. The most popular type of DoS or distributed DoS (DDoS) attack is flooding, which simply bombards a victim with packets beyond the victim's bandwidth or processing capacity. For instance, the DDoS attack on root DNS servers on October 21, 2002 mobilized 100 to 200 kilo-packets per second against each victim, blasting 50 to 100 Mb/s of traffic on it [1]. Such degree of abuse shoves the victim into a state where it can hardly respond to any legitimate request. Figure 1a shows a typical DDoS scenario, in which the attacker hides behind masters that compromise agents and let them attack the victim. As for a worm epidemic, it manifests itself in the form of *hostscan* in terms of network traffic. In an attempt to infect other hosts, it scans a wide range of IP addresses by sending packets to them (Fig. 1b). Hostscan can also be used by hackers to probe hosts, usually for a single vulnerability. Somewhat less important is *portscan,* in which the attacker probes open vulnerable ports (i.e., services) on a single victim machine. As we saw in recent episodes, DoS attacks and global worm epidemics such as Code Red [2], Nimda, and SQL Slammer [3], incur huge economic and social costs. These attacks must be detected in their early phase and blocked as much as possible.

Detecting suspicious network activities and providing early warning to network administrators is an essential yet difficult task. Especially when we go deeper into the network, the speed and the sheer volume of legitimate traffic make the task of pinpointing ongoing attacks look practically impossible. However, in this article we demonstrate the feasibility of devising such a method that simultaneously detects, calibrates, and visualizes multiple ongoing attacks in real-time with great precision.

Attack Visualization

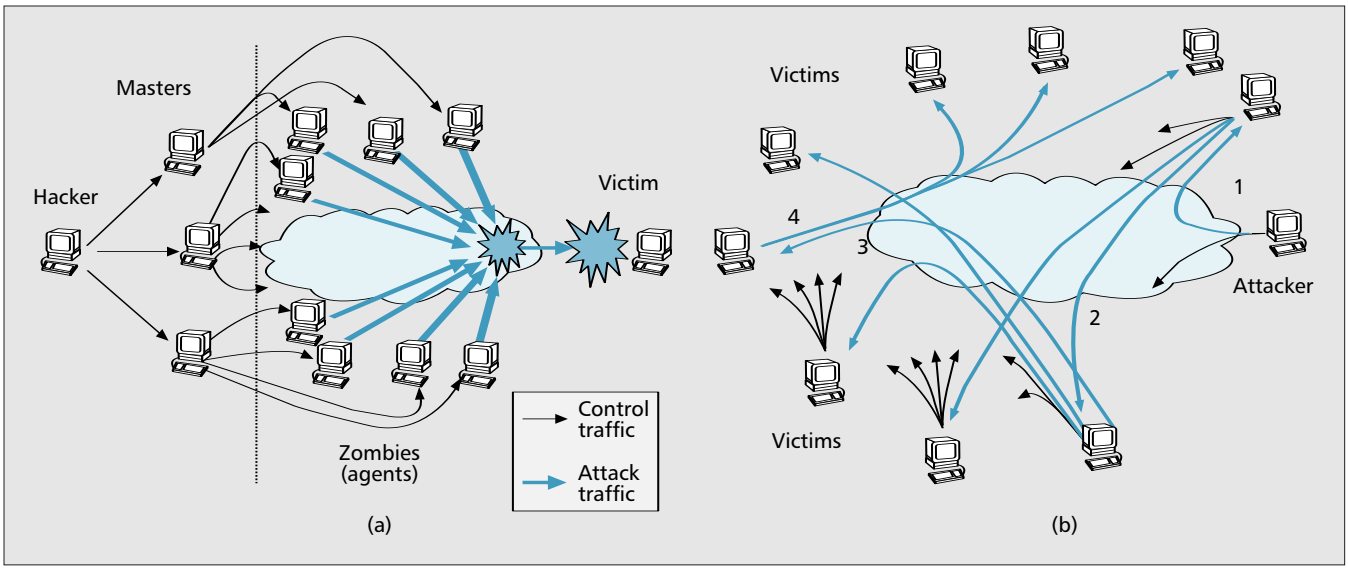
Our idea for intuitive attack visualization is simple: use three header values available from most Internet packets, namely, the source IP address (I_s), the destination IP address (I_d), and the destination port number (P_d). Using these three values as the dimensions of a Cartesian space, we map each packet into the three-dimensional space according to their values. For a real example, we show a visualization of packets from 90Mb/s trans-pacific backbone traffic in Fig. 2 (for convenience, IP addresses are in decimal). This figure is plotting 2.2 million packets captured in an 85-second interval on one typical day in December 2001.

Each dot in the figure represents a "flow," which in this article is defined as a set of packets sharing the same I_s, I_d, and P_d (e.g., the packets in the same TCP connection).

What is surprising in the figure is that unusually regular geometric formations are found, striking out from an amorphous background. These geometric figures are, indeed, the very manifestations of attacks. First, the rectangles parallel to $I_s - P_d$ plane are source-spoofed DoS attacks. We can see one in the front, one in the back lying low, and barely visible in front and up in the high port range. If the destination port is not randomly filled, source-spoofed DoS attacks would appear as a line parallel to the source IP address. Second, we can see five distinct lines parallel to the destination IP address. These are hostscans. (The two thick lines are in fact many lines.) Third, the lines parallel to the destination port are portscans. The schematic diagram in Fig. 3 summarizes the observation.

Why do attacks manifest themselves in such a visibly regular formation? The answer is threefold. First, in general, legitimate traffic is widely and irregularly dispersed in the three-dimensional space. The size of the three-dimensional space is huge — $2^{32} \times 2^{32} \times 2^{16}$ — but the number of concurrent legitimate flows is expected to be much less even in backbones (typically less than one million) [4]. Therefore they appear as amorphous and thin clouds. Second, in stark contrast each *attack packet constitutes a separate flow under our definition*, and it makes each attack stream look highly inflated. While a single dot collectively represents all the packets in legitimate flows, there are as many dots as there are packets

0890-8044/04/$20.00 © 2004 IEEE



■ Figure 1. *a) Distributed DoS attack by flooding; (b) worm epidemic.*

in the attack flow. So even if there are a vastly larger number of legitimate flows than attacks, attack flows strike out, as we can clearly see in Fig. 2. (Notice how conspicuous a single DoS attack flow, DoS1, is against the background.) In the case of DoS, this spurious flow explosion is due to its *spoofing* maneuver. It is well known that most DoS attacks randomly fill the source IP address in the IP header in order to obfuscate traceback efforts [5]. Other header fields can also be randomly filled along with the source IP address, as long as the destination IP address is correctly aimed at the DoS victim. So $I_s = *$, $I_d = I_v$, $P_d = [P_v | *]$ where the subscript v represents the victim and "*" denotes a wildcard. In Fig. 2 we can see that DoS1 utilizes only half the source IP space and the entire destination port range, whereas DoS2 uses the entire source address space and only well known ports (< 1024).

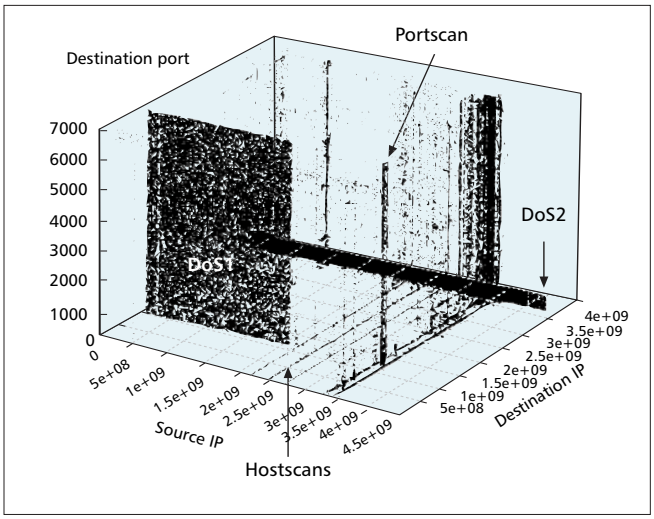
In Fig. 4a we show a filtered real-life packet trace that exemplifies the source spoofing behavior in a DoS attack stream (for privacy reason we "sanitized" the victim's address). We can see that port numbers are being randomly filled, which would result in the rectangle formation in the three-dimensional visualization.

In the case of host scanning, the scanner (a scanning program or a worm) does not know a priori which address is in use and vulnerable, so it has to blindly scan the IP address space. This random address generation across a wide range results in the line formation. In Fig. 2 hostscans all apparently scour the entire IP address range except the Class D and E spaces. The mode of search varies depending on the implementation. Figure 4b shows a real-life example of a hostscan (again, the scanner's identity is masked for privacy). In the case of port scans, skimming through the port space inevitably leaves the line mark parallel to the destination port axis. Interested readers can find a sample minute-by-minute animation of visualized attack activities from our Web site [6].

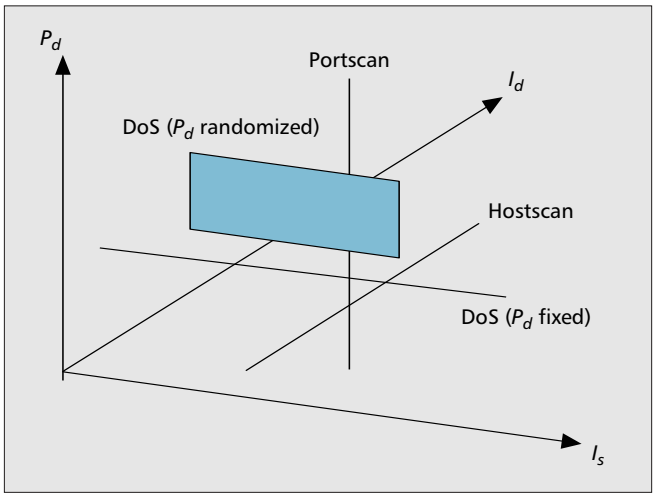
System Behavior in Non-Backbone Environments

Our visualization method works best in backbone environment where there are no particular prominent traffic sources or destinations. However, if we use the method in an environment where a dominant traffic source or sink is near, care should be taken since legitimate traffic can also form regular geometry. An example is a popular Web server with IP address I_w in an enterprise network. A large number of requests can be launched from all over the IP address space toward I_w. If we visualize the traffic at the boundary of the enterprise network, the incoming request traffic can form a DoS-like line with $I_s = *$, $I_d = I_w$, $P_d = 80$. Note that only DoS has this ambiguity problem, and hostscan and portscan do not.

One way to address the DoS ambiguity problem in edge



■ Figure 2. *3-d plotting of a trans-pacific traffic.*



■ Figure 3. *Schematic diagram of attacks.*

Time	Source address	Source port	Destination address	Destination port
.....	.....	.....	.....	.....
09:35:03.319081	67.171.49.204	7804	x.x.x.x	16675
09:35:03.319647	20.214.51.196	47582	x.x.x.x	16675
09:35:03.319652	55.44.55.180	61602	x.x.x.x	16687
09:35:03.319922	55.44.55.180	61602	x.x.x.x	16687
09:35:03.320607	89.130.59.164	10086	x.x.x.x	16695
09:35:03.321665	56.184.129.14	4787	x.x.x.x	16706
09:35:03.322084	117.152.194.136	51005	x.x.x.x	16709
09:35:03.322098	3.164.5.250	5928	x.x.x.x	16716
.....	.....	.....	.....	.....
09:35:03.325331	25.123.15.210	8210	x.x.x.x	16736
09:35:03.326188	6.188.152.174	23371	x.x.x.x	16754
09:35:03.326565	6.188.152.174	23371	x.x.x.x	16754
09:35:03.327048	87.231.154.166	63149	x.x.x.x	16768
09:35:03.327248	101.242.222.24	18073	x.x.x.x	16765
.....	.....	.....	.....	.....

(a)

Time	Source address	Source port	Destination address	Destination port
.....	.....	.....	.....	.....
09:35:23.955222	x.x.x.x	64218	72.142.101.184	111
09:35:23.958716	x.x.x.x	64232	72.142.101.197	111
09:35:23.965132	x.x.x.x	64310	197.14.58.120	111
09:35:23.965443	x.x.x.x	64311	197.14.58.121	111
09:35:23.966412	x.x.x.x	64316	197.14.58.126	111
09:35:23.974520	x.x.x.x	64322	197.14.58.132	111
09:35:23.976617	x.x.x.x	64331	197.14.58.141	111
09:35:24.091332	x.x.x.x	64424	19.231.216.127	111
09:35:24.093271	x.x.x.x	64423	19.231.216.126	111
.....	.....	.....	.....	.....
09:35:24.104956	x.x.x.x	64438	19.231.216.141	111
09:35:24.105238	x.x.x.x	64437	19.231.216.140	111
09:35:24.106191	x.x.x.x	64433	19.231.216.136	111
09:35:24.107471	x.x.x.x	64429	19.231.216.132	111
09:35:24.125654	x.x.x.x	64466	85.114.173.117	111
.....	.....	.....	.....	.....

(b)

■ Figure 4. *a) DoS attack trace; (b) Hostscan trace.*

networks could be combining an address validity check with our geometry-based algorithm. Below we discuss two address validity check schemes. Note that not only do they reduce the DoS ambiguity at edge networks, they also help ascertain the hostscan and portscan attempts more quickly. Also, they are applicable in backbone networks as well.

Exploiting the Global Address Allocation Map — A significant portion of the IP address space is still unallocated by IANA (Internet Assigned Numbers Authority) [7] or designated as "Martian" [8]. The use of such illegal addresses in a flow strongly suggests that the flow is an attack [9]. The barcode-like figure in Fig. 5 shows the IP address allocation map as of Nov. 2003. For one example, there is a big unallocated chunk in the latter half of the Class A space (see arrow). Since no packet can originate from or be destined to these spaces, any regular geometric formation rendered by legitimate traffic, if any, should be lacking points in the spaces. In the figure the first example formation (dotted line in the middle) has points matching the unallocated space. Such line formation can only be left by an attack stream. Likewise, DoS1, DoS2, and hostscans in Fig. 2 have points corresponding to the unallocated spaces, so they cannot be due to legitimate traffic. The second example (dotted line at the bottom), on the other hand, lacks points in the unallocated space, so it is likely that this is legitimate traffic.

Using the Local Address Assignment Information — Another visual confirmation of an attack can be obtained from the fact that either the source address or the destination address should be from the downstream network of the observation point. For instance, if the monitor is situated at the boundary of an enterprise network, the packets coming into and going out of the network should have the destination address and the source address assigned to the enterprise network, respectively. If the observation point is at an ISP, at least one of the addresses should be that of its subscriber network. In Fig. 2, for instance, we are certain that DoS1 is an attack since its source address spans the entire class A space. This should not happen since the downstream network of the particular vantage point providing the view of Fig. 2 happens to be the Korean Internet, which has only a small fraction of class A.

Figure 6 shows how these two techniques can be used to confirm the attacks that have been identified by our geometry-based algorithm. The figure is a two-dimensional projection of Fig. 2 onto the $I_s - I_d$ plane. Solid arrows mark DoS attacks evidenced by their illegitimate positions in terms of the local allocation. The dotted arrows mark the attacks that are so classified due to the global IANA-unallocated address usage, as depicted in Fig. 5.

Automatic Extraction

The three-dimensional visualization method discussed above can certainly provide network administrators with "eye candy" — intuitively recognizable signs of ongoing attacks. However, simply mapping each and every arriving packet onto the three-dimensional space has practical problems. While plotting every packet in the three-dimensional space in real time is already difficult, this approach has the more important limitation that low intensity attacks and the attacks in the proximity of dominant legitimate traffic may not be visible enough in the three-dimensional plot. In order to present only meaningful (i.e., attack) information to the network administrators, extracting the attacks from legitimate background traffic is crucial. Therefore from now on we will focus on how to extract only the attack information from the high-speed packet flow in real-time.

The idea of the extraction algorithm starts from the observation from Fig. 4: attacks fast "pivot" one or more values of the flow information. In hostscan, for instance, I_s and P_d are fixed, while I_d pivots on them. In portscan, P_d pivots on I_s and I_d. In source-spoofed DoS, I_d is fixed, while either only I_s or both I_s and P_d pivots on it. Instead of employing complex (thus slow) pattern recognition techniques such as three-dimensional edge detection, we apply an original algorithm that captures the "pivoted movement" in one or more of the three coordinates. This is because, from a graphical perspective, such movement forms the aforementioned regular geometry along the axis of the pivoted dimension(s).

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 three binary values: $<K_s, K_d, K_p>$. The coordinates in the signature correspond one-to-one to the flow coordinates. Each coordinate value in the signature tells us whether the corresponding

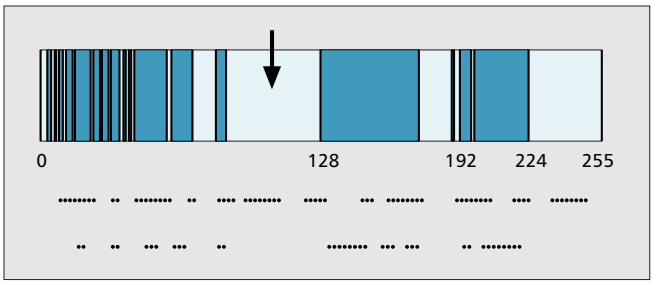


Figure 5. *IPv4 address allocation vs. example line formations.*

value in the flow (to which the packet in hand belongs) was seen "recently" or not. (The degree of recentness for different coordinates could vary, and we will deal with it later.) For example, suppose the following two flows pass through the monitor that executes our scheme.

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

For convenience, throughout the article we will call the monitor *RADAR monitor* (for Real-time Attack Detection And Report); the algorithm that it executes we will call the *RADAR algorithm*. The RADAR remembers these two flows for a finite time duration L. For the sake of explanation we will assume for now that the time duration is the same for every coordinate, for example, $L = 2$. When a packet with source IP = 1.2.3.4, destination IP = 3.4.5.6, destination port = 90 appears at time $t + 2$, the RADAR determines that this packet's signature is $<K_s, K_d, K_p> = <1, 0, 1>$. 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 in either flow (1) or flow (2) as the destination address, so K_d = "0". If $L = 1$, flow (1) would have been purged from the 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. In Fig. 4b, for instance, the pivoted coordinate is I_d, and each packet presents a new value: 72.142.101.84 → 72.142.101.197 → 197.14.58.120 → … . Hence the RADAR will keep generating the <1, 0, 1> signature for hostscan. In this manner the RADAR yields 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 a 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 for another, all due to hapless modification of one or more coordinates in the signature. For instance, in Fig. 4b if a previously unobserved legitimate flow destined to 197.14.58.121 passes through the RADAR immediately after this hostscan, it would get "1" in the destination address coordinate. The accuracy of the proposed algorithm thus depends on the likelihood of these unwanted changes in the signature. The analysis of the statistical aspect of the algorithm is rather tedious, so it is omitted here. But the conclusion of our study [6] is that unwanted transformations can be suppressed by careful treatment of L and other simple techniques.

Attack Signatures

Based on the above idea Table 1 exhaustively enumerates all attack 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 lists the number of dimensions that are pivoting. The second column lists how the attacks might manifest themselves geometrically when the attack is mapped onto the three-dimensional space as in Fig. 2.

Most of the signatures in Table 1 are fairly straightforward, but there are three that we reject as unlikely on a technical basis [6] (shaded in the table). Also, distributed DoS (DDoS) does not appear in Table 1. We can consider two cases. If DDoS sources spoof a source IP address, they will collectively be detected as a single DoS attack <0, 1, *>. If spoofing is not used, since individual DoS streams resemble legitimate flows from the RADAR's viewpoint, they will not be detected as an attack. Usually, however, DDoS mobilizes a large DoS network of agent hosts to maximize the impact. For example, more than 359,000 machines were made an agent by Code Red version 2 [2] in an attempt to bombard the White House Web site. Therefore when the attack commences the RADAR will suddenly begin to see a great many source IP addresses. This will produce a noticeable amount of <0, 1, *> signatures at a fast pace, and draw the attention of the RADAR. Although this is not an *intended* operation of the RADAR, DDoS attacks with no spoofing will trigger an alert provided

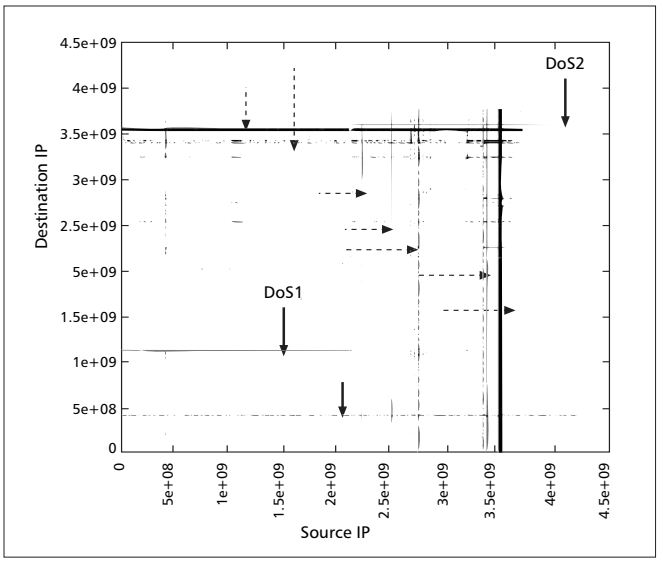
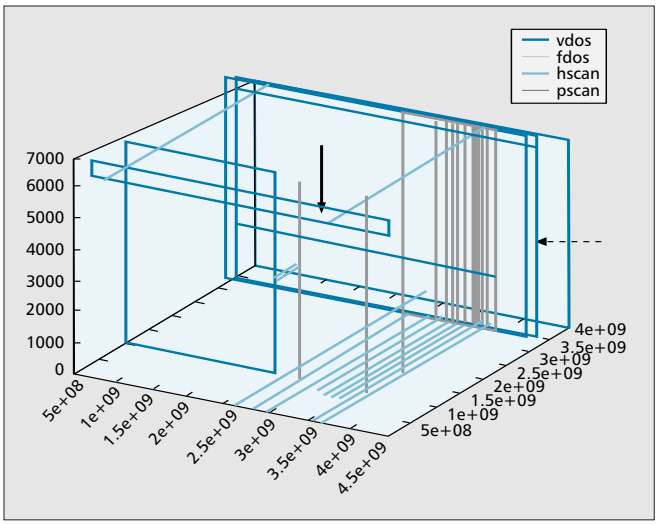


Figure 6. *2-d projection of Fig. 2 onto* $I_s - I_d$ *plane*

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

Table 1. *Attack signatures.*



■ Figure 7. *Presenting only the attacks embedded in Fig. 2.*

the intensity exceeds the threshold used on a spoofed DoS attack from a single attacker. Finally, DDoS or DoS attacks without spoofing, although there are few [5], can be detected by the traditional IDS method of packet counting, and can be entirely blocked and even traced back. Therefore we do not treat the non-spoofing DoS scenario in this article.

The remaining five cases are of practical importance. First, "Kamikaze" is odd since a single source spews packets at a high rate toward random destination hosts at random ports. Apparently it cannot be an effective attack, and it seems rather suicidal. The origin of this type of "attack" is not clear, but it does appear in our traces [6]. One explanation could be a bug in the DoS attack code — pivoting on the destination address instead of on the source. But a more plausible theory is that it is the backscatter [10] from the DoS victim toward spoofed attack sources. Next in Table 1 we list two DoS types, but the distinction is only for the convenience of analysis — it does not bear any practical significance. Finally, in case hostscan searches for several vulnerabilities (e.g., represented by port numbers p_1, p_2, ... p_n) at the same time, the RADAR is designed to deal with the attack as n separate hostscans (albeit perpetrated by the same source). Namely, the fixed port number assumption still applies to individual component hostscans.

Legitimate Signatures

Having discussed the attack signatures, we now ask ourselves what will *legitimate* traffic receive as the signature? In principle the signature should be either <0, 0, 0> or <1, 1, 1>. The former is given to the first packet in the flow, since the RADAR sees this flow for the first time. For the subsequent packets in the same flow the latter should be the signature since the s, d, p values have already been observed in the first packet. Fortunately, these two are not likely signatures of attacks in Table 1. Therefore, legitimate packets ideally get past the RADAR with little obstruction, only three hash table lookups. No further analysis is necessary.

Although our study shows that the chance of unwanted signature modification in real-life traffic is thin and even controllable to a large degree [6], occasional misinterpretation can occur. As discussed above, when an attack signature is generated the RADAR is supposed to check the per-source (per-destination) frequency in the back-end. With only incidental signature modifications, therefore, there is little chance for a legitimate flow to accumulate enough flow counts and eventually lead to a false positive. This is in stark contrast with attacks that for their inherent needs (e.g., maximizing impact in DoS or speeding up the worm's spread) rapidly accumulate the count.

Execution Result

Figure 7 shows the results of applying the RADAR algorithm to Fig. 2. We set the attack thresholds at 5/s for scans and 50/s for DoS. The setting is intentionally low to put the RADAR's sensitivity to the test. In the figure 'vdos' means DoS with P_d =*, while "fdos" means P_d is fixed (which was apparently not found). We notice that the faint DoS attack of Fig. 2 is clearly classified here (solid arrow). DoS2 is reported to occupy larger space than the real attack (dotted arrow). This is due to an outlier flow(s) that happens to lie on the same plane as DoS2 but at a high port range. The current implementation does not address this minor problem yet. Also, we find that more hostscans are now visible, much more than the prominent five in Fig. 2.

Figure 7 was created from textual reports on the attacks, two of which can be confirmed in the trace as shown in Fig. 8 (privacy masking still applies). The report in Fig. 8a tells us that the attack has a rate of 1,277 pkts/s. The corresponding trace in Fig. 8b tells us that it is a 40-byte TCP SYN flood attack ("6" = IP protocol number, "2" = TCP flag). The spoofed source address range (0.55.237.128 to 128.61.29.32) and the source port range (1 to 65534) verifies that this is DoS1 of Fig. 2.

The report in Fig. 8c signals a hostscan for the secure shell port (22). It starts from 71 network and ends at 115 network. Currently the RADAR does not graphically represent the intensity of a given attack, so this strongest hostscan in the

34572.24 (0.55.237.128, 128.61.29.32) -> 66.x.x.x (1, 65534) 60985 47.755 (1277.03)

(a)

093612244565	56.189.204.205	12339	66.x.x.x	35366	40	6	2
093612247255	30.51.143.67	45678	66.x.x.x	35371	40	6	2
093612254666	31.191.216.157	54399	66.x.x.x	35390	40	6	2
093612254936	31.191.216.157	54399	66.x.x.x	35390	40	6	2
093612258342	86.96.157.11	61980	66.x.x.x	35399	40	6	2
.....							
093622455306	117.179.241.193	36834	66.x.x.x	18399	40	6	2
093622456673	11.138.251.153	39116	66.x.x.x	18419	40	6	2
093622460791	2.36.148.53	44821	66.x.x.x	18469	40	6	2

(b)

34523.80 210.x.x.x -> (71.118.0.4, 115.83.255.253) 22 10935 96.201 (113.67)

(c)

093600023828	210.x.x.x	22	115.83.154.172	22	40	6	2
093600025149	210.x.x.x	22	115.83.154.178	22	40	6	2
093600026286	210.x.x.x	22	115.83.154.180	22	40	6	2
093600026885	210.x.x.x	22	115.83.154.183	22	40	6	2
093600028027	210.x.x.x	22	115.83.154.187	22	40	6	2
.....							
093622556421	210.x.x.x	22	115.83.247.64	22	40	6	2
093622556493	210.x.x.x	22	115.83.247.66	22	40	6	2
093622557695	210.x.x.x	22	115.83.247.71	22	40	6	2

(d)

■ Figure 8. *a) TCP SYN flooding indication by RADAR; b) TCP SYN flooding in the trace (in part); c) Hostscan on ssh port indication by RADAR; d) Hostscan on* `ssh` *port in the trace (in part).*

textual output is not easily seen in Fig. 7. This is going to be addressed in our future work.

Another trace-based test appears in Fig. 9. Figure 9a is the result of the RADAR's analysis for a one-minute trace from a campus network in 2003. The RADAR reports two hostscans, and from the trace we confirm that the scanning sources are Code Red II worms. The 3818-byte payload is one of its signatures (Fig. 9b), but we also found that these identified sources performed more than 1500 SYN infection attempts during the one-minute interval (most of which are futile, as expected).

Implementation

Figure 10a shows the construction of the *front end* of the RADAR monitor, called "the main filter." It is composed of three hash tables, and collectively these hash tables generate the signature for each incoming packet. A single, separate lookup is made against the source IP address, the destination IP address, and the destination port number table, respectively. When a value (address or port) is "not found," that is, recently unobserved, it is *registered* in the corresponding hash table as a new sighting. Any hash function can be used as long as it has good distributional properties and can be quickly calculated. Among these two properties, however, the speed is more important for the front end. For instance, MD5 and SHA-1 may have good distributional properties, but the computation they require is too complicated, so they would not fit our environment. Our experience shows that using the least significant 24 bits from the IP address suffices for casual operation. Against the backbone trace we have, it resulted in 1.0072 comparisons, on average (most are 0 and 1, where 0 means an empty hash bucket), with only a few reaching up to eight comparisons. For the port hash table the hash function is identity, because there are only 64K port number values. Since the hash lookups are used, the complexity of the main filter can be engineered at O(1).

The reason we perform a separate lookup for each coordinate should be clear. If we maintained each flow entry indexed by $<s, d, p>$ collectively, we would not know which coordinate is responsible for a failed flow lookup. Thus we would not know immediately which coordinate is being pivoted, that is, what type of attack is being mounted. Then some additional processing would be necessary on these new flows in order to achieve classification. Therefore, for real-time classification separate hash lookups are essential.

Hash Table Entry Lifetimes in the Front End

Associated with each entry is the last accessed time t_{last}. We maintain a moving time window L beyond which registered IP addresses or port numbers age out. Namely, if $t_{now} - L > t_{last}$, we remove the entry from the corresponding hash table. We call the time window a *lifetime*, and we use two lifetimes as follows:

- L_H: the lifetime for I_s and I_d.
- L_p: the lifetime for P_d.

L_H must have a larger value than the inter-packet gap for most flows. If not, the address value in every packet in many flows will be considered new, generate "0" in the corresponding signature coordinate, and the flow will be suspected of performing pivoting in that coordinate. This is about the only guideline in the configuration of L_H, and we can set it to a fairly large value without any substantial adverse effects [4]. The reason is simple: the only packet in a legitimate flow that could be adversely affected by too large L_H is the first packet. This single packet signature transformation from "0" to "1" in an address coordinate will only push the flow count by one in the back end. Therefore the probability of false positive alarm

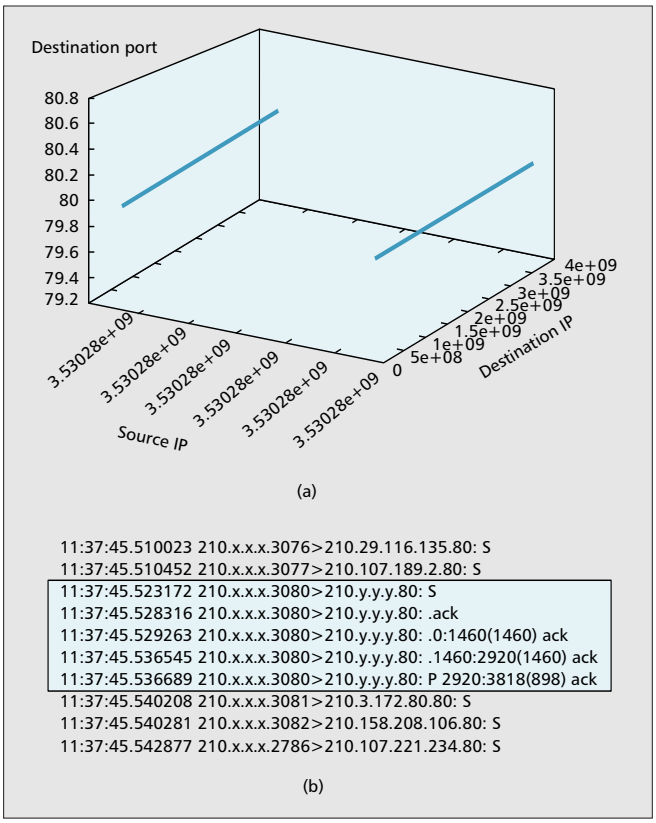
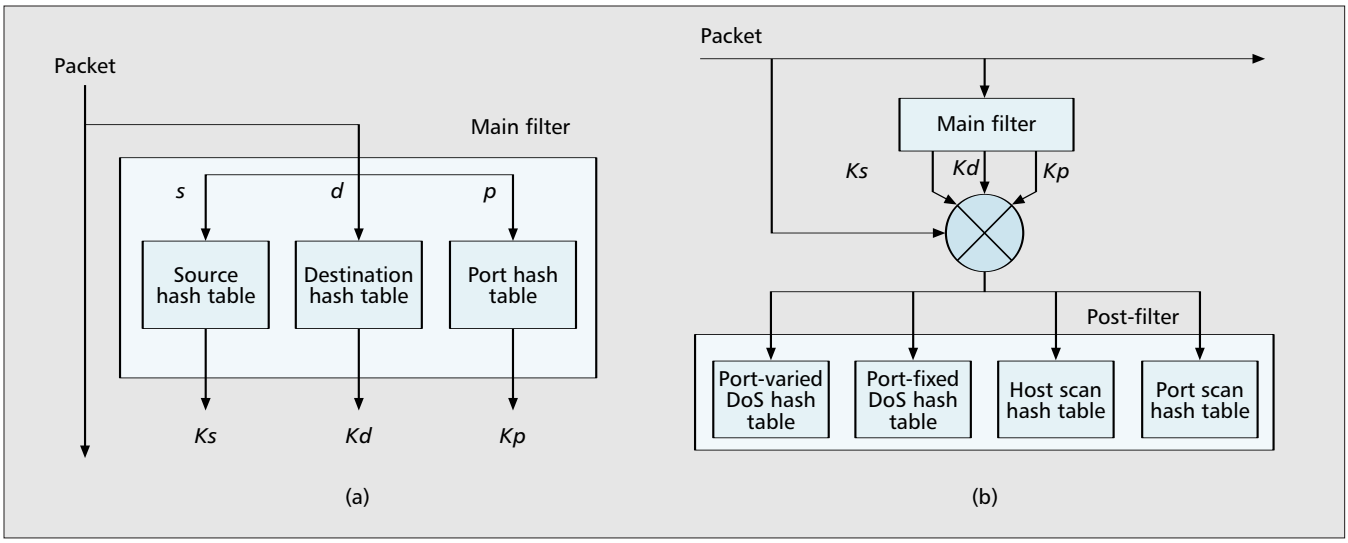


Figure 9. *a) RADAR graphical report of hostscans; b). code Red II in action (tcpdump, in part).*

in the back end is low. Likewise the false negative probability is also low since attacks pivoting an address coordinate will encounter a relatively small number of active and legitimately used host addresses during address pivoting. The IP address space is vast and the probability of incidental "0" to "1" transformations is relatively low. The interesting scenario is when there are multiple address pivoting attacks. Since each of the attacks registers numerous addresses in the monitor, the probability of encounters between different attacks at an address may not be negligible. Suppose there are A concurrent attacks with V pivots (scans or spoofs) per second each. Then the number of host address entries will be $AV L_H$. When an attack randomly picks an address for the next pivoted address, the probability of picking up an already registered value (that leads to "0" → "1" transformation) will be roughly $AV L_H/2^{32}$. For example, if there are 1,000 concurrent attacks with 10,000 pivots per second each, and L_H = 10s, the transformation probability is approximately 1/400. This seems tolerably low for such a large number of high-intensity attacks, but the precise selection should depend on the value of A, V, and the target probability of tolerable signature transformation. In our current implementation we use the value of 10s. As to L_p, we have a different situation. The port number space is only 64K, and the chance of "0" → "1" transformation under a large L_p is not negligible. But fortunately it has little impact on the detection of more important, global attacks such as DoS or hostscan (by a worm or by a scanner) because the hostscan signature already has "1" in the port number coordinate, and the port-varied DoS attack <0, 1, 0> being transformed to a port-fixed DoS attack <0, 1, 1> is of no practical concern. However, it can boost the false negative rate for portscans (i.e., <1, 1, 0> → <1, 1, 1>). If one wants to reduce the false negative rate for portscans, a small L_p should be used. It has no adverse effect on detecting DoS because, again, <0, 1, 1> → <0, 1, 0> is a DoS-to-DoS transformation. For hostscans,



■ Figure 10. *a) Signature generation by the front end ("main filter"); b) back end ("post filter") and the whole RADAR monitor organization.*

however, the <1, 0, 1> → <1, 0, 0> transformation can cause a false negative problem in the back end. To prevent this from happening it should be $L_p \geq 1/r_{hs}$, where r_{hs} is the hostscan detection threshold in the back end. Such L_p will not cause a "1" → "0" transformation in the destination port coordinate for the hostscans that need to be detected. To meet such constraints the minimum L_p that we can have is $1/r_{hs}$. In our implementation we use 0.2s for L_p.

When we purge stale entries (with $t_{now} - L > t_{last}$) we may choose one of two approaches. First, we can periodically clear them altogether. Second, we can embed the purge operation in the lookup procedure. We take the latter approach in our current implementation, since the front end requires uninterrupted operation. Specifically, in the matching operation in the hash tables we first check if the entry under comparison is stale. If so, we purge the entry before continuing the matching. This requires one additional comparison for each entry for a timestamp check, and a list update operation upon encountering each stale entry. With the example of the source address table, we can roughly compute the probability of encountering a stale entry in a lookup operation. Suppose N is the number of new source addresses registered every second. With the same lifetime applied to the entries, stale entries are generated asymptotically at the same rate. If there are B buckets in the hash table each bucket will generate N/B stale entries per second. When K packets pass the RADAR monitor in a second, each bucket will be subject to K/B lookups per second, so each lookup (i.e. arriving packet) encounters a stale entry with a probability of N/K. Since usually $K >> N$, the probability is fairly small and so is the per-packet cost of removing stale entries. In the worst case all arriving packets can be attack packets, creating a new entry. Then $K \sim N$ and each lookup will accompany a single purge operation, which is the upper bound for the overhead.

Back End Operation

Figure 10b shows the whole structure of the RADAR. In addition to the main filter we have discussed so far, there is a back end called "the post filter." The main filter performs the detection and preliminary classification while the post filter verifies if the classification is correct and measures the attack. Table 2 shows the minimum information maintained at each post-filter module.

Each post-filter table is also a hash table, indexed by either the destination IP hash or the source IP hash. The design is modular enough, so we could add or remove analyzer modules. Also, each module can be augmented with other more complex functions than we describe here. For one important example, if suddenly a large number of hostscans on the same port begin to be reported and the host scanning packets carry the same payload, the hostscan analyzer might decide to look into the payload for a worm using a virus scanner.

As shown in the table the post filter has enough information to report either who the attacker is (scans) or who the victim is (DoS). It also maintains the packet count which, combined with the duration of flow activity since it was "booked" in the post filter ($t_{last} - t_{init}$), is used to estimate the average attack rate that in turn is compared with the detection threshold. Here, t_{last} is the time instance of the tracked flow's last recorded packet arrival. As to the DoS and hostscan detection thresholds, we can drastically lower them by augmenting the analyzers with the address checker that we discussed earlier. Specifically, if a suspected attack is found to use illegitimate addresses we could declare the attack earlier even before it reaches the normal threshold.

Other values are used to calibrate the dimension(s) of the attack: the range of spoofed addresses and port number(s) (DoS), the range of scanned IP addresses and the port num-

Analyzer	DoS (fixed)	DoS (varied)	Hostscan	Portscan
Indexed by	*d* (victim)		*s* (perpetrator)	
t_{init}	•	•	•	
t_{last}	•	•	•	•
s_{min}	•	•		
s_{max}	•	•		
d_{min}			•	•
d_{max}			•	
p_{min}	•	•	•	•
p_{max}		•		•
Pkt count	•	•	•	•

■ Table 2. *Calibration information maintained at analyzers.*

ber (hostscan), or the scanned address and the range of scanned port numbers (portscan). These are only used for generating coordinates in the graphical representation of the attack. Without having to spray hundreds of thousands of dots on the three-dimensional plot, we use this trimmed down representation in summarizing attacks to the administrator, thereby saving memory and drastically reducing the plotting time. This is how Fig. 7 was generated.

The partition of the RADAR into a front end and a back end fits better with the high-speed link environment for two reasons. First, we have an extremely tight time budget and so cannot afford to apply a per-packet examination thorough enough to reach a final judgment. Second, even in the face of the most intense attacks, the majority of the traffic is still legitimate. Thoroughly examining *all* packets is inefficient; the front end needs to pan out only suspicious packets through simple signature tests, passing the legitimate packets with the least obstructions. Only suspicious packets matching an attack signature are handed over to the back end for closer examination.

Evaluation

Field Test

We plugged the unoptimized, prototype RADAR into the campus network gateway at Seoul National University. The incoming packets were optically tapped from the gateway router on two Gigabit Ethernet interfaces [6]. A Pentium-4 2.4GHz machine with 512MB Rambus memory, Intel PRO/1000MF dual port LAN card, and PCI 2.2 (32bit) bus simultaneously run a separate instance of the RADAR algorithm on each Ethernet port. The total traffic rate was roughly 330Mb/s (65Kp/s) at the time of the experiments. The algorithm is lightweight enough so that there was no packet loss at the kernel [6].

Simulation

Except for the speed test, the field trial against live traffic is hardly helpful. Since we do not know a priori how much attack is contained in the traffic, we cannot test the RADAR in terms of sensitivity, precision, and false positive rates. So for the evaluation in a more controlled environment we resorted to simulation. The simulation methodology goes as follows. First synthesize background traffic and inject a prescribed attack therein. Next the resulting contaminated traffic runs through the RADAR, which pans out the attack. Finally, compare the attack calibration reported by the RADAR monitor with the attack prescription. For the synthetic background traffic the address and port distributions and flow arrival process were statistically modeled [6]. By default the parameters are set as follows: L_H = 10s, L_p = 0.2s, $r_{hs} = r_{ps}$ = 5/s, r_{dos} = 50/s. The flow arrival rate is set to λ = 10,000/s. The detection thresholds are intentionally set extremely low in order to put the sensitivity of the RADAR algorithm to the test.

Now we define the performance metrics. First *sensitivity* is defined to be the ratio of detected attack instances to injected instances. Suppose the attack exceeding the threshold persisted for M observation intervals. If the RADAR monitor is sensitive enough it should have picked up the attack in all M intervals. But if it detected it in only D_t intervals the sensitivity S is given by:

$$S = \frac{D_t}{M}$$

Second, the *relative error* E is given as:

$$E = \left| \frac{\lambda_M - \lambda_D}{\lambda_M} \right|$$

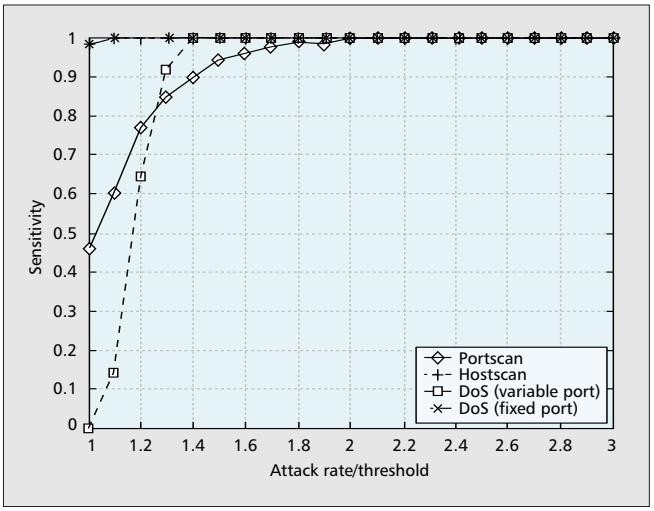


Figure 11. *Sensitivity, default setting.*

where λ_M is the mounted attack intensity and λ_D is the detected attack intensity. For instance, if the attack rate is 100/s and the detected rate is 120 (or 80) the relative error is 0.2. Obviously the relative error is measured only on detected attacks. Third, the *false positive rate* P_f is given by:

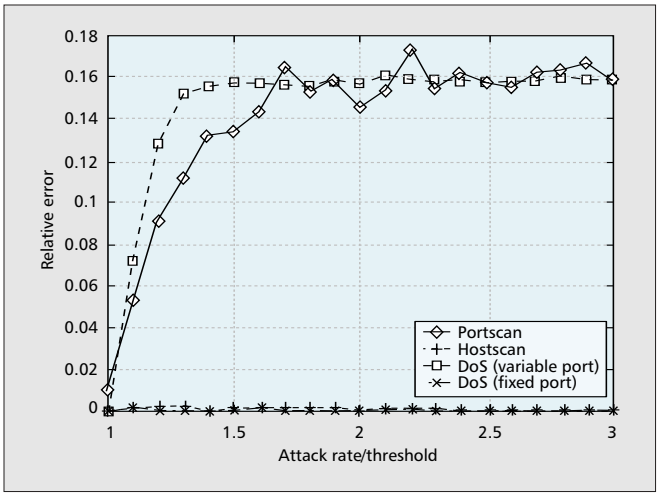
$$P_f = \frac{D_f}{D_t + D_f}$$

where D_f is the number of instances of attacks that are not injected (i.e., they are from the background traffic).

Sensitivity — Figure 11a shows the sensitivity in the default setting. The horizontal axis is the ratio of the attack rate to the threshold for each attack type. The figure shows that the RADAR algorithm is extremely sensitive to hostscan and port-fixed DoS, while portscan and port-varied DoS detection is more disrupted by the background traffic. This is understandable since for portscan, the <1, 1, 0> → <1, 1, 1> transformation probability is quite high for L_p = 0.2s. Likewise, for port-varied DoS, the <0, 1, 0> → <0, 1, 1> transformation probability is high, but recall that the distinction between the two DoS types is only for the sake of analysis. If we count together the port-varied DoS (i.e., the original) and port-fixed DoS (i.e., the transformation) the sensitivity problem disappears [6].

Relative Error — Figure 12 shows the relative error for the default setting. Again, portscan and port-varied DoS suffer from the port signature transformation problem. Portscan is not a global threat we are interested in here, so we are not overly concerned. Again, for the port-varied DoS, when counted together with the port-fixed DoS it has a near-zero relative error [6].

False Positive Rate — We used unrealistically low thresholds for the scans and DoS above merely to demonstrate that the RADAR is sensitive and accurate enough to pinpoint even very low-intensity attacks. In this experiment, however, we attempt to make them a little more practical by multiplying them by five. Namely, we set $r_{hs} = r_{ps}$ = 25/s, r_{dos} = 250/s. Considering the average rate of 4,000 scans/s per infected host in SQL Slammer, the scan threshold of 25 is still only miniscule. As to DoS, the aggregated attack rate that we assume here still cannot constitute an effective attack on any major target. In terms of bandwidth, it amounts to 160 kb/s to 480 kb/s, assuming TCP SYN flood. In contrast, the DDoS attack on root DNS servers mobilized 100 to 200 Kilo-packets



■ Figure 12. *Relative error, default setting.*

per second per victim (root server), blasting 50 to 100 Mb/s of traffic on it [1]. The parameters in our simulations are set very low simply to emphasize the high sensitivity and low error and false positive rate of the RADAR algorithm. Note that when we scale the thresholds the attack intensity is also proportionally scaled up. Throughout these experiments L_H remains at 10s, and only L_p is set to the reciprocal of the scan thresholds, as we discussed earlier.

Figure 13 shows that the false positive rate quickly approaches near zero values for all types of attacks as the attack intensity goes over the thresholds by a factor of 1.2. We still see the effect of port signature transformation in variable-port DoS and portscan, but it is quickly overcome as the attack intensifies. Although we do not show the result, we also tested 10-fold scaling of the thresholds, where the false positive rate drops to 0 for all attack types from the beginning.

Memory Requirement

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 2^{25} hash buckets whose heads are a pointer (usually four octets). This alone is 128MB. Over and above that we need to store each flow in these tables, where a flow has at least two IP addresses, one 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 one million flows being tracked simultaneously, 34MB should be used. Then one million flows in the main filter IP table translates to approximately 10Gb/s (OC-192) based on our flow arrival rate constant, since we have by default L_H = 10s. Over and above that we have the port table in the main filter. However, there are only 64K entries, thus it adds little to the memory requirement. In the post filter we do not have large tables since concurrent attacks must be small in number. We do not expect to see, for example, 64,000 attacks all simultaneously under way even it is on a backbone link. Therefore we use 16-bit hash for all tables. Again, the memory requirement will be insignificant, most likely less than 2MB. In sum, more than half the memory of the RADAR is used to construct the IP tables in the main filter. If memory is a critical resource we could use 23-bit hash, halving the requirement, and then 22-bit hash, etc. But on a high-speed link we assume that speed is the primary object, not memory.

Related Work

Visualization

There is an array of academic and commercial work underway on DoS detection and countermeasure, as well as initial attempts at modeling worm spreading dynamics [11, 12]. As for attack visualization, however, there is a dearth of tools, not to mention those that are specifically focused on intuitive presentation of ongoing attacks in real time. Following is a short list of related work, including both academic and commercial packages.

There is an open source project called the Shoki Packet Hustler that uses similar two-dimensional or three-dimensional visualization techniques as ours [13]. It was originally written as a diagnostic widget for a network intrusion detection system (NIDS). Currently the tool implements only the graphical front end for manual analysis of IP network data. The addition of automatic analysis codes is planned.

FlowScan [14] analyzes NetFlow data and can provide visualization in the units of five-minute intervals. It can plot flow and packet count by IP protocol, port number, IP prefixes, or AS pairs. Given this functionality, it can reveal a DoS attack by an abrupt increase in the number of flows unaccompanied by an increase in bandwidth usage or packet count. (There is no mention in the original FlowScan paper as to whether it can detect hostscans or portscans based on the flow count.) One concern with FlowScan is that under attack its stateful inspection feature suffers from the explosion of flow entries since the flow export rate is equal to the packet forwarding rate, as pointed out in our early discussion.

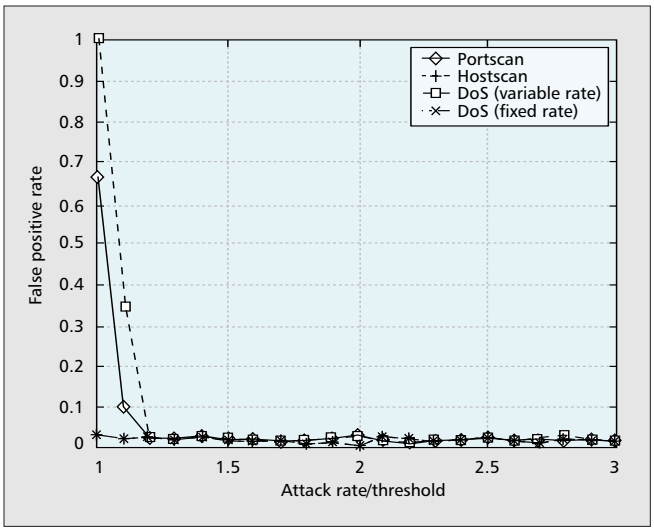
Estan *et al*. propose a new method of traffic characterization that automatically and dynamically groups traffic into minimal clusters of conspicuous consumption [15]. The automatic grouping is a means of balancing insufficient dimensionality and excessive detail, thus making the traffic data more amenable to the network administrator's consumption. This functionality can prove useful when there is a pressing need to understand and respond to sudden traffic spikes such as network worms or DoS attacks, being able to identify malicious aggregates. This can be achieved by using so called multidimensional clustering, through which multidimensional cluster plots are created.

Although not focused on visualizing attacks the Etherape tool [16] distinguishes itself from other tools such as tcpdump and Ethereal in that it graphically reports network traffic.

There are also commercial products that provide graphical representation of network traffic. However, even when they do they are not specialized for attacks. Rather, attacks are supposed to be inferred by network administrators from traffic behavior aberrations reported by the tool. Mazu Network's Profiler™ [17] has graphical profiling capability for network traffic in terms of IP addresses, protocols, ports, and flow volume. The profiling depicts noticeable events and relative anomalies rather than attacks per se. The detection of such events and anomalies is done every 60 seconds, based on the latest statistics, so there is much latency. OpenService's Security Threat Manager™ [18] can show noticeable events or correlations of events as a topology. The graphical view is available as an interactive chart, and it contains the summary of involved IP addresses (source and target), observer's IP address, attack category, port numbers, and correlation factor. The reporting time frame ranges from minutes to months. Peakflow™ is an anomaly detection tool from Arbor Networks [19]. Among its capabilties it can build granular, dynamic graphs that display traffic and routing information, both real-time and historical, by AS, router, interface, and protocol.

IDS vs. RADAR

Since the functionality of an IDS is much broader, it would be unfair to put IDSs and the RADAR in direct comparison. We will only discuss how traditional IDSs and the RADAR are related. The RADAR is composed of the front end and the



■ Figure 13. *False positive rates.*

back end. In a sense what the RADAR performs in the back end can overlap with what IDS does, but it would be more accurate to say that the back-end behavior is intentionally underdefined (except for visualization). So the back-end functionality can range from simple calibration to complex antivirus scans against those packets that were classified as suspicious by the front end. However, the RADAR front end is not what a traditional IDS can simply replace. First of all, IDSs are difficult to deploy on very high-speed links, with their heavy rule base and complex matching requirements. More importantly the separate *s*/*d*/*p* lookup (not counting) enables a single RADAR front end to simultaneously track the characteristic movement of DoS, hostscan, and portscan in the graphical sense. In contrast, traditional IDSs *count* packets. For instance, a high packet count from a particular source could indicate the possibility of a hostscan. However, this inevitably leads to a high false positive rate. Lacking the capability to exactly characterize and filter out legitimate packets from the count, the IDS is likely to lose sensitivity on attack behavior, so an innocent but sufficiently active source or destination can be misjudged to be hostscanning or under DoS attack, respectively. On the other hand, if the IDS counts each *s*-*d* or *s*-*d*-*p* tuple, the false positive problem can be mitigated, and an attack can be identified by a sudden explosion of the tuple count. However, it will sharply increase the required number of observed flow entries due to the increased dimension(s). Such multi-dimensional flow table explosion and consequent lookup penalty are exactly what the RADAR tries to avoid by checking the packet against *s*, *d*, *p* separately in the front end.

Summary

Network attacks such as DoS and scans manifest themselves as a regular geometric entity in a three-dimensional space whose dimensions are source IP address, destination IP address, and destination port number. These attacks appear highly inflated (i.e., occupying a visibly significant portion) in this three-dimensional space due to the inherent need of evasion maneuver (in case of DoS) or wide scanning in a short amount of time (host/port scanning), whereas legitimate flows only occupy a single point. Thus this representation can visually assist network operators to easily recognize ongoing attacks.

One technical issue with the representation, though, is that it is not scalable and in some cases provides poor discrimination between attacks and ambient legitimate traffic. Hence there is a need to extract only the embedded attacks and present them while maintaining the intuitive grace of the original presentation. Instead of employing complex pattern recognition algorithms to detect such regular patterns, our RADAR algorithm captures the "pivoting" behavior of attack packets, which directly translates to the forming of the abovementioned regular geometry in the three-dimensional space. The RADAR algorithm requires only a few memory lookups per packet, yet the classification error is minimal. This algorithm pans out only suspicious packets matching the pivoting behavior, buying enough time for a more sophisticated back-end processing that significantly reduces the false positives. The simulation and real implementation experiments have been done, and the algorithm indeed performs up to our expectation on high-speed links.

Acknowledgments

The original idea of classifying the attacks using the three-packet header values is attributed to Heejung Sohn. We thank Jin-Ho Kim and Byung-Seung Kim for their effort in running the RADAR code on live Seoul National University backbone network traffic. We also thank the anonymous reviewers who helped improve the presentation of the article. This work was supported in part by a Korea University Grant and a NRL program of KISTEP, Korea.

References

[1] P. Vixie (ISC), G. Sneeringer, and M. Schleifer, "Events of 21-Oct-2002," http://www.isc.org/index.pl?/ops/f-root/october21.txt, Nov. 2002.
[2] CAIDA, "CAIDA Analysis of Code Red," http://www.caida.org/analysis/security/code-red/coderedv2_analysis.xml, July 2001.
[3] CAIDA, "Analysis of the Sapphire Worm," http://www.caida.org/analysis/security/sapphire/, Jan. 30, 2003.
[4] IP Monitoring Project at Sprint, http://ipmon.sprint.com/ipmon.php
[5] K. Houle and G. Weaver, "Trends in Denial of Service Technology," CERT Coordination Center, Oct. 2001; available at http://www.cert.org/archive/pdf/DoS_trends.pdf
[6] 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
[7] IANA, "Internet Protocol v4 Address Space," http://www.iana.org/assignments/ipv4-address-space
[8] F. Baker, "Requirements for IP Version 4 Routers," RFC 1812, June 1995.
[9] H. Kim and I. Kang, "On the Effectiveness of Martian Address Filtering and its Extensions," *Proc. IEEE GLOBECOM 2003*, Dec. 2003.
[10] D. Moore, G. Voelker, and S. Savage, "Inferring Internet Denial-of-Service Activity," *Proc. 2001 USENIX Security Symp.*
[11] M. Garetto, G. Gong, and D. Towsley, "Modeling Malware Spreading Dynamics," *Proc. IEEE INFOCOM 2003*, Mar. 2003.
[12] D. Moore *et al.*, "Internet Quarantine: Requirements for Containing Self-Propagating Code," *Proc. IEEE INFOCOM 2003*, Mar. 2003.
[13] http://shoki.sourceforge.net/hustler/.
[14] D. Plonka, "Flowscan: A Network Traffic Flow Reporting and Visualization Tool," *Proc. 2000 USENIX LISA.*
[15] C. Estan, S. Savage, and Varghese, "Automatically Inferring Patterns of Resource Consumption in Network Traffic," *Proc. ACM Sigcomm 2003*, pp. 137–48.
[16] Etherape, http://etherape.sourceforge.net
[17] Mazu Network Profiler, http://www.mazunetworks.com
[18] Open Network SecurityThreatManager, http://www.open.com
[19] Arbor Networks Peakflow, http://www.arbornetworks.com

Biographies

HYOGON KIM (hyogon@korea.ac.kr) has been an associate professor at Korea University, Seoul, Korea since 2003. From 1996 to 1999 he was a research scientist at Bell Communications Research, Morristown, New Jersey. From 1999 to 2003 he was an assistant professor at Ajou University, Korea. His research interests include Internet protocols and architecture, and wireless communication.

INHYE KANG is an assistant professor at the University of Seoul. She received the Ph.D. from the University of Pennsylvania. Before joining the University of Seoul she was a visiting professor at Soongsil University from 1998 to 2000, and a chief engineer at Samsung SECUi.com from 2000 to 2002. Her research interests include software engineering, formal methods, and Internet security.

SAEWOONG BAHK received the B.S. and M.S. degrees in electrical engineering from Seoul National University in 1984 and 1986, respectively, and the Ph.D. from the University of Pennsylvania in 1991. From 1991 through 1994 he was