Towards Tracing Hidden Attackers on Untrusted IP Networks *

Ho-Yen Chang, S. Felix Wu

{hchang2, wu}@eos.ncsu.edu
2300 Research IV, Centennial Campus
North Carolina State University, Raleigh, NC 27695
Tel: (919) 515-7920, 513-1894, Fax: (919) 513-1895

C. Sargor, X. Wu

{sargor, xwu}@picard.mcnc.org
Advanced Networking Research, MCNC
RTP, NC 27709
Tel: (919) 248-1852

Abstract

Recent denial-of-service (DoS) attack has highlighted the need to quickly and efficiently trace the sources of the attacks. In particular, tracing attacks with "spoofed" source IP addresses has been proved to be extremely difficult so far. In DECIDUOUS project described here, we present a security management framework for identifying the "true" sources of network-based intrusions. The premise of this approach is that *if an attack packet has been correctly authenticated by a certain router, the attack packet must have transmitted through that router.* It utilizes IPSec security associations to do dynamic deployment of secure authentication tunnels and further traces down the possible attackers' locations. In this paper, we first review some related research work on this area and then present the architecture and algorithms to support the tracing of multiple attacks launched from different locations, even across several administrative domains. Our result shows that the DECIDUOUS system is reasonably efficient, flexible in supporting different tracing strategies, robust against certain attacks/failures directly on DECIDUOUS itself, and correct in determining the true sources of attacks even when the attacks are coordinated. This approach could serve as the basis for future research on different tracing strategies for different type of attacks in large-scale networks.

Keywords: Network Control and Security Management, Tracing Denial-of-Service Attacks, Dynamic Authentication Tunnel, IPSEC Security Association

1 Introduction

Recently, the network security community has made reasonably good progress in the development of attack prevention and intrusion detection technologies for protecting hosts against network-based attacks launched remotely by an intruder. However, because of the design of *Internet Protocol* (IP), identifying the source in real-time of such attacks remains relatively difficult.

The current IP network only provides an unreliable, connectionless, best-effort packet delivery service, so datagrams delivered may be lost, duplicated, delayed or out of order. Furthermore, it is difficult to eliminate forged IP packets or find their true sources in a large, high-speed network due to the stateless nature of Internet routing. Similar to mail service, the intermediate routers only use packet's destination address to make routing decisions, and source address is used only when the reply from the destination is necessary. This property has given hackers a lot of space to play without any penalty. One serious attack, for instance, denial-of-service (DoS) attack, takes the advantage of forged IP packets and becomes the mainstream of attack incidence.

DoS attacks prevent the targeted site from providing network services by either flooding the site with bogus packets, consuming finite network resources, slowing down the services which need real-time constraints such as online auction, stock transactions, or even causing the site to crash [9]. For instance, many sophisticated distributed denial-of-service attacks (DDoS) were launched in February of 2000 against major e-commerce Internet sites. Hackers jammed a Web site with lots of useless messages that tied up the site's computers and slowed the site's operations for hours [8]. Unless we have a good defense strategy, Denial-of-service will remain to be one of the most difficult challenges in terms of securing the Internet that we face.

Tracking down the sources of a flood-type DoS attacks may be even harder. A victim today may only rely on tools not intended for this purpose, such as `traceroute` and `finger`, which are only suitable

*This research project is supported by DARPA/ITO.

for tracing an unsophisticated attacker using his real source IP address. Another big challenge in attack source identification is to track down the intruder's location across multiple administrative domains. A simple example will be to identify an attack source from another RNP (Regional Network Provider) or ISP (Internet Service Provider). Today, a case like this will usually involve a lot of human interventions. By the time system administrators from different organizations start to cooperate, a sophisticated intruder may have escaped either logically or physically.

The objective of the Decentralized Source Identification System (*DECIDUOUS*) [4, 5] project is to securely, practically and systematically identify attack sources by utilizing, as much as possible, existing network security protocols and services and to avoid introducing any new protocol under a single administrative domain. Specifically, the IPSec authentication and encryption services [19, 20, 21] are used by the security management module to trace the source of an attack. The DECIDUOUS system deduces the source identification information from the end-point locations of the current security associations in the attacking packets, and this leads to two key concepts in DECIDUOUS:

1. Every "established" IPSec security association that has been or has not been used by the attack packet can provide some location information about the true sources;

2. A set of security associations should be strategically established as "trapdoors" such that the collected location information will be sufficient to decide the true sources.

In other words, DECIDUOUS will use a Dynamic Security Association Deployment algorithm to **dynamically decide** *where, when and how to establish security associations through IPSEC/IKE* in order to efficiently identify the attack sources. In this paper, we present the intra-domain and inter-domain tracing algorithms for single and multiple attackers.

This paper first introduces different classes of network attacks and the traceback problem in Section 2. We review some existing approaches for DoS attacks in Section 3 and describe the concept of authentication tunnels and the state-of-art of IPSec technology with dynamic security associations in Section 4. Section 5 provides an overview of DECIDUOUS and introduces the concept of *CutSet* topology translation. It also describes the algorithms for intra-domain tracing and multiple attackers' tracing. Section 6 provides a collaboration protocol for inter-domain tracing. Section 7 presents the design and implementation of the DECIDUOUS framework. Experimental results are given in Section 7.1. Finally, the conclusions and future work of this study are presented in Section 8.

2 Network-based Intrusions, Attacks

Network-based intrusions can be classified into three different types: *simple attacks*, *network infrastructure/service attacks*, and *compound attacks*. First, for simple network-based intrusions (*e.g.,* SYN-Flood), an intruder merely *uses the available network services* to launch attack packets against some victim connected to the Internet. Second, an intruder/insider can directly attack the network infrastructure itself to disrupt or maliciously control the network services. A good example is the MAI router incidence in April, 1997, where a faulty BGP router de-aggregated and advertised thousands of network addresses which caused much of the Internet to be disconnected from 20 minutes to 3 hours. The MAI incidence implies that potentially an intruder can selectively de-aggregate certain routes to absorb/intercept all the traffic destined to those target networks. In our lab environment, we have successfully demonstrated that, as long as we can intercept OSPF packets coming-in and going-out of a good router, we can convert this good OSPF router into a very malicious OSPF router [32, 18, 33, 6] without replacing the router kernel software (such as CISCO's IOS). Third, a sophisticated intruder can first attack the network infrastructure, and then use the compromised/controlled network services to attack some other victim hosts. Furthermore, immediately after the latter attack, the intruder can "*restore*" the network infrastructure services back to normal. Under such orchestrated attacks, existing networking utilities like *router filtering* or *traceroute* will not be able to effectively stop the attacks or identify the attack source(s).

Currently, network based intrusions are not handled systematically, and usually human system administrators must be heavily involved. Typically, a system administrator, after hearing some complaints over the phone or emails, needs to spend a significant amount of time to check various log files. He then might use existing utilities like *traceroute* or *tcpdump*. If the suspected attack source is in another domain, he most likely will make a few more phone calls to get some help from the administrators in another organization. Even after all these efforts, it is still not always possible that the true attacker will be identified.

2.1 Hit-and-Run and Persistent Attacks

Network-based attacks can be either *persistent* or *hit-and-run* [32]. *hit-and-run attacks* such as the WinNuke attack, the Ping 'o Death attack, and recently the TCP-Land attack require only one bad packet to either crash or seriously slow down a victim system, although they may use multiple packets to keep a machine down. On the other hand, for *persistent attacks*, the intruder must frequently launch attack packets to invade the victims. For example, flooding attack needs constant and rapid transmission of packets to the victim in order to be effective. Because attacker can forge the source address of the flooding packets without reducing the effectiveness of the attack, it is non-trivial to determine the true origin of such attacks.

Heuristically, the *attack source* for a hit-and-run attack is hard to identify. By the time the damage caused by such attacks is discovered, the attacker has hidden itself by restoring the normal behavior on the compromised entities. Certainly, one possible way to catch such attacks is to turn on all the intrusion detection or auditing options all the time on all possible network entities (routers, subnets, and hosts). This approach is obviously very expensive and usually slows down the network performance even when no attack exists. Persistent attacks could be launched by a *single attacker* or *multiple attackers*. If it is by a single attacker, although the source address of the attack packets are spoofed, the attacker's physical location can not be changed during the attack. Otherwise it is a variant of hit-and-run attack. It is not uncommon for the intruder to simultaneously control many entities distributed over the whole network system, or even worse, across different administrative domains. The attacker can coordinate all the compromised entities to collaboratively launch attack packets. The attack source(s) for the latter case, "*i.e., a coordinated attack,*" is relatively harder to identify and isolate. DDoS attacks belong to this category. The attackers broke into several Internet computers (usually attached to high speed nets) and installed attack agents. The attackers then used these distributed agents to launch simultaneous DoS attacks against a targeted site. Because attacks come from many domains at the same time and high level traffic generated by the attack overwhelms filtering and tracing tools, it increases the difficulties in fighting DDoS attacks.

2.2 Resource Allocation and Denial of Service Attacks

Several efforts [12, 34, 25, 23] have been put into the formal specification and verification on DoS. The discussion was to focus on the loss of availability due to accidental or malicious user actions, as opposed to random failures that impact functionality, which are the concern of reliability engineering. Because long delays in service constitute denial, and absolute denial can be viewed as an infinite delay, a DoS protection is to limit waiting time for access to some resources. That is, regardless its load at any time, a system should guarantee a maximum response time for certain specific actions [25]. Yu and Gligor [12, 34] characterized the DoS problem as how to provide a shared service with a specific maximum waiting time between users' competitions. Their approach introduced the concept of a "finite waiting time policy" which is implemented through a combination of "user agreements", fairness and simultaneity polices, and service specifications, and is stated by temporal logic formulas such that whenever a service has been requested, it will eventually be provided. Extending the Yu-Gligor resource allocator work, Millen [25] proposed a Denial-of-Service Protection Base (DPB), using *probabilistic waiting-time policies* and *resource monitor* stated in the context of a state-transition model, to guarantee that the authorized access to resources is under its control, and no any process should hold one resource so long that no other process can get progress.

All of these works assumed that the attacker cannot spoof its identity within the same system, so the resource holders can be easily identified, and the resource allocator can decide who should be revoked. However, it may not be applied to network environments where no one has a trusted identity. Since DoS attacks may occur even before parties are authenticated to each other, it is difficult to rely on the enforcement of proper access control policies to protect us. Meadows [23] presented a formal framework and evaluation method to show some principles that can be used to make protocols more resistant to network denial of service such that we can build good defenses as much as possible into the protocols themselves. Meadows later also described some open issues in the formal methods for protocol analysis in [24].

One major security problem of TCP/IP design is that there is no unique identity attached with each machine, and IP address just a logic concept which can be configured to any thing at any time. Furthermore, in mobile service, IP address is not even a string attached with geological location. Thus, when we review the historical lessons from several DoS attacks, there is one common point: *systems need* *to provide fair service to every one but cannot verify the identity and determine the fair share of resources.* There is no *source accountability* in the current Internet.

2.3 Traceback problem

Definition 2.1 (Attack Path) *Let $G = (V, E)$ be an undirected graph and v_t is the target victim node to be protected. $A_i \in V$ is a potential attacker which will launch one or more attack packets to v_t through some routers $R_i \in V$. An* **attack path** *is an ordered list of nodes along which the attack packets traveled between A_i and v_t. An* **attack period** *p_{att} is the minimum length of time interval that the attack need to execute to successfully bring the victim down.*

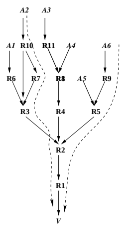

Figure 1: Upstream router map from victim

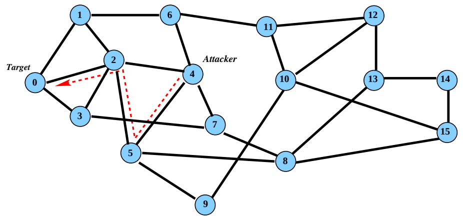

Figure 2: An example of the attack path traceback

The victim V could be a single host under attack or a network border device such as a proxy firewall for many hosts. As shown in Figure 1, if we represent the upstream routers from V into a tree map, the leaves $\{A_i\}$ are the potential attack origins or attackers. The *attack graph* is the graph composed of the attack paths, e.g., two attack paths, $(A_2, R_{10}, R_7, R_3, R_2, R_1, V)$ and $(A_6, R_9, R_5, R_2, R_1, V)$. We refer to the packets used in DoS attacks as the attack packets. Within a true attack path, we call the first edge **attack origin edge** that carries the attack packets, e.g., edges (A_2, R_{10}) and (A_6, R_9) here. The other edges could only be used to relay the packets. If the attack origin edge could be found, the attack path can also be easily derived by using tools such as traceroute.

Definition 2.2 (Traceback Problem) *The* **traceback problem** *is to find the attack path and the associated attack origin for each attacker. The real-time online traceback is to find the attackers within some* **trace period** *p_{trace} such that $p_{trace} << p_{att}$.*

The efficiency of a trace system can be evaluated based on the length of the trace time (i.e., trace period p_{trace}), the number of attack packets to be observed, and the resource of network to be used.

For example, in Figure 2, R_0 is the victim and R_4 is the attacker which keep sending large amount of attack packets to R_0 through the attack path (R_4, R_5, R_2, R_0) (instead of the shortest path (R_4, R_2, R_0)). To find an exact attack path is generally complicated and the true attack origin may never be revealed because of the spoofed source address. All the hosts hooked up with this compromised router could be suspects. Generally, we assume that each normal router will choose a shortest route with least hops to forward packets, and the routing is fairly stable within the attack period. For example, if only R_4 is compromised, the possible attack paths that R_4 could use to attack are (R_4, R_6, R_1, R_0), (R_4, R_5, R_2, R_0), (R_4, R_7, R_3, R_0), and (R_4, R_2, R_0). As we can imagine, the more neighbors the attacker has, the more possible attack paths it can manipulate, and the more difficult to trace. Note that a router may control the traffic from its upstream routers but not its downstream routers. In this example, if R_4 is the attacker, the attack packets must pass through at least one node in the set $\{R_4, R_5, R_6, R_7\}$ and then at least one node in the set $\{R_3, R_2, R_1\}$.

Practically, we may restrict the problem as finding a candidate ordered list which contain the attack path as a partial ordered set of nodes. For example, $(A_i, R_6, R_5, R_4, R_3, R_1, v_t)$ and $(A_i, R_6, R_4, R_3, R_2, R_1, v_t)$ are both candidate paths cover the exact attack path $(A_i, R_6, R_3, R_1, v_t)$. Moreover, *it may not be necessary to find the whole attack path.* Ideally, the goal is *to find a router as close to the real attacker's origin as possible*, e.g., R_6 here, or the corresponding attack origin edge (A_i, R_6), and to do some security management actions from there. The DECIDUOUS algorithm presented later can trace down the attack source to a small enough region or even the compromised link (i.e., attack origin edge) used by the attacker.

3 Related Work

There have been several efforts proposed [31, 11, 4, 3, 30, 27, 2, 28] to prevent and trace IP spoofing attacks and [9] provides a good survey on this work.

3.1 Ingress and Egress filtering

One way to eliminate the ability of forging source addresses is to install *ingress and egress filtering* [11, 3] on every router such that packets with illegitimate source address will be blocked. If the attacker was forced to only create outbound connections using the legal address space, sooner or later some administrator would know where is the problem. ORNL [10] provides a "spoof-tester program" that would allow a user to verify if his ISP is using proper ingress filters. ICSA NetLitmus [16] provides a tool that can test whether or not the ISP's filters are actually working.

This approach is not effective unless all the ISPs agree to implement and deploy universally. How to unambiguously distinguish between the normal and illegitimate traffic is a big challenge. As traffic is aggregated from multiple ISPs, it is usually difficult to determine the legitimate of a packet source. Furthermore, comparing each packet against a filter list in a very high speed is also a challenge. For performance issues, backbone routers and edge routers for complex topologies probably could not be configured with such filters. Even with ingress or egress filtering widespread deployments, the attacker could still use many spoofed addresses from thousands of hosts within a valid network for a considerable attack period without being caught.

3.2 Hop-by-hop Input Debugging

In hop-by-hop input testing, the victim's intrusion detection system (IDS) first develops a *attack signature* to describe the common feature of this attack for all the attack packets and gives this signature to a network administrator to install a input debugging filter on the victim's upstream egress port to determine which ingress port coming from and which upstream router originated the traffic. Once determining the next router to test, one logs into that router to determine next one to be tested, and so forth. This tracing procedure is performed manually or via some automated tools and is repeated recursively on the upstream routers until the attack source is reached. CISCO [7] described how to use CISCO router and set up access control list (ACL) to characterize and trace packet floods. Stone [30] at UUNET proposed an Overlay Network approach, *CenterTrack*, by using IP tunnels to reroutes the interesting attack flow from edge routers directly to some instrumented tracking routers. The tracking routers can easily determine the ingress edge router by observing which tunnel the datagrams arrive on. Van's thesis [31] presented an Active Network to build and dynamically deploy a filter to filter out forged packets within the network, thereby foiling SYN-Flooding attack.

Similar to a phone call tracing, this approach assumes that an attack remains active until the completion of a interactive trace. However, the considerable management overhead may slow down the tracing process and result in an incomplete task. For tracing across different domains, this task is even more difficult because of the political issues or private administrative rules.

3.3 Probabilistic Marking and Sampling

Clearly, the simplest way to do traceback is like the IP Record Route option to append each intermediate router's address to the end of the packet as it travels through the network. The victim can then know the attack path without any extra work. However, this approach has infeasible high router overhead and can lead to unnecessary fragmentation, because the path could be potentially very long such that the packet size exceeds MTU. To reduce the overhead, Savage [27] proposed a clever encoding algorithm to let all the intermediate routers probabilistically mark in-flight packets with partial path information as they arrive at routers. Because attacks generally compromise large numbers of packets, each marked packet will represent a "sample" of the path it has traversed. A victim then can reconstruct the complete path after receiving a modest number of such packets. This approach allows victims to identify the approximate source of an attack without the assistance of outside network operators, and no extra traffic is generated. The trace information is bound to the packets in the same path, so will not be blocked by fire-walls or policy routing mechanisms. Moreover, this trace information can be logged and used to performed post-mortem analysis after an attack has completed.

This approach needs to reserve some additional space in IP header to record the path information and is not backward compatible. For a large amount of packets, this overhead is expensive and may lead to fragmentation. To reduce per packet storage requirements, they proposed a technique to compress the information as a hash and overload it into 16-bit IP Identification field used for fragmentation in the IP header, based on the recent argument that only less than 0.25% of packets are fragmented [29]. However, the ID field of IP header should not be changed if fragmentation is necessary, but their marking procedure can violate this property. IPSec AH [20] provides cryptographic protection for the ID field; if it is modified, the packet will be discarded by the receiving system. Furthermore, to produce the total router order from the sample distribution is not only a slow process but also an easier trap lead to misorder path unless huge amount of data need to be collected. When under serious flooding attacks, even the sample itself could get lost due to the high packet dropping rate which will incur the incorrect distribution information and plausible-looking fake attack paths could happen. Both false positive and false negative could occur at this case.

3.4 Sampling by ICMP Traceback Message

Bellovin [2] proposed that when forwarding packets, routers can randomly generate a new ICMP traceback message (ITRACE) with a low probability (about $1/20,000$) along the path and sent to the destination. Each router ITRACE message contains information fields about*back link on the previous hop, forward link on the next hop, timestamp, traced packet, and authentication for preventing fake Traceback messages.* With enough ITRACE messages from enough routers along the path, the traffic source and path can be determined.

Although the additional traffic seems small to a net increase at the destination, this approach may introduce extra traffic into network, which could worsen the already congested links under the flooding attacks. In order not to increase the traffic too much, the probability must be small enough. However, under a serious flooding attacks, ITRACE message itself could also get dropped due to its low probability of existence which will disturb the distribution and result in an incorrect inference. Furthermore, because ITRACE message may not follow the same route as the attack packets it traced, it is unclear this approach can survive the DDoS attacks which involved a massive amount of traffic from many different sources and different paths, and the plausible-looking fake chains may deceive a victim to frame another innocent victim. Note that both marking sampling and ITRACE sampling approaches described above involved protocol implementation changes in routers. Their effectiveness require to modify Internet routers to ensure the proper deployment and backward compatible with the existing infrastructure. Moreover, because the probability is fixed for each router, it is possible that attacker can carefully control the attack packet rate dynamically such that the victim cannot collect enough samples within a short period of time to construct the correct path.

4 Dynamic Authentication Tunnel and IPSec Security Associations

The attack source identification module in DECIDUOUS requires *network topology information*, which can be provided by either the network administrators or routing daemons. Furthermore, the router policy information on the routers is useful in optimizing the identification speed. In this paper, we **will not** discuss both issues (*i.e.,* the security of network topology information and the router's policy) and only concentrate on algorithm design and implementation parts.

4.1 Dynamic Partial Authentication Graph

Definition 4.1 (Communication Channels and Authentication Graph) *Let $G =< V, E >$ is a directed graph, where V is the set of parties in the network and E describes the communication channels. That is, for any $u, v \in V$, $< u, v > \in E$ if and only if there is a communication channel between u and v.*

If some pairs of parties in V can form some secret **authentication channels**, *either by the secret-key system (one key) or the public-key system (two keys), then $G_A =< V_A, E_A >$ is an* **authentication graph** *[1], where E_A is the set of the authentication channels, $V_A \subseteq V$ is the set of parties which can do authentication through some $e_A \in E_A$.*

Authentication channel is formed by using each node's secret key. That is, if a message sent through an authentication channel $< u, v >_A$, an intermediate attacker can *only drop or damage the message but cannot modify the original message or forge an new message using u's signature*, because he did not know the key. Note that authentication channel is a logical concept. It can be realized by a physical link connection or a virtual tunnel which is much cheaper implementation. The computational complexity of authentication and verification depends on the authentication and encryption algorithms. It is usually required to be robust enough to resist man-in-the-middle attack.

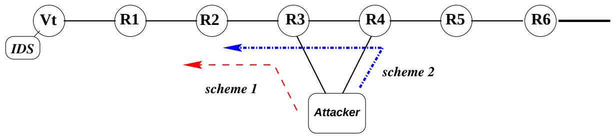

Figure 3: Attacker on a simple topology

Consider a simple attack example on a linear network topology as in Figure 3. Internet routing tables usually remain stable over long periods of time and routers are assumed to know correct routes. Assume that all the routers can do authentication for the packets destined to node v_t, and with correct network topology information, a good router should always forward the packets through the shortest path. A compromised router can choose to drop the packet, to authenticate the packet, or not to authenticate the packet, but it cannot fake the other router's signature. Within an autonomous domain of a finite size, let $\mathcal{R}$ be the set of all the routers and $\gamma_i^d \in \mathcal{R}$ is a router γ_i with d hops away from v_t. For example, γ_3^3 is a router with 3 hops away from v_t. Let $authVer(pkt_{att}, n_i)$ be a boolean function of the intrusion detection system (IDS) in v_t to verify the correct authentication of a given node n_i in the given packet pkt_{att} and Γ is the set of routers which authenticate this packet correctly. That is,

$$\Gamma = \{\gamma_i^d | authVer(pkt_{att}, \gamma_i^d) = true, \forall \gamma_i^d \in \mathcal{R}\},$$

and φ is the set containing routers which do not authenticate this packet properly.

Given an attack packet pkt_{att}, the attacker's location then can be determined by a region $[\gamma_i^{d_1}, \gamma_j^{d_2}]$ consisting of two routers, γ_i and γ_j, with d_1 and d_2 hops away from v_t respectively, such that $d_1 < d_2$ and

$$d_1 = max\{d | \forall \gamma_i^d \in \Gamma\} - 1$$

$$d_2 = min\{d | \forall \gamma_j^d \in \varphi\}$$

For example, if an attacker is at the location between R_3 and R_4, he can send the attack packets with spoofed source addresses through two possible ways to victim v_t: via path $\{R_3, R_2, R_1, v_t\}$ or via path $\{R_4, R_3, R_2, R_1, v_t\}$. If attacker is using the first attack scheme, relaying packets through router R_3 first, the attack packets will be authenticated by $\{R_1, R_2, R_3\}$, and his location can be determined by a region $[R_2^2, R_4^4]$; otherwise, the attack packets will be authenticated by $\{R_1, R_2, R_3, R_4\}$, and his location can be nailed down to the region $[R_3^3, R_5^5]$. Furthermore, by investigating the authentication interface address, it is possible to nail down exactly which link has been compromised if none of the routers are compromised. That is, the link address can be known as the first router interface to authenticate with largest hop number away from victim. On the other hand, DECIDUOUS will not nail down exactly which router or link has been compromised if some routers or security gateways themselves are compromised. In the latter case, it will identify the attack source as an small area containing a few routers and links in general.

If all the communication channels are authenticated (*i.e.*, $E_A = E$ and $G = G_A$), we call it a *complete authentication graph*. Because all the messages are authenticated, so it is impossible to forge any communication through any channel. Thus it is totally trusted. However, complete authentication graph in practice is too costly to realize, and it degrades the performance of the network. In DECIDUOUS, we use *dynamic partial authentication graph* to approach the attacker's location without raise any attention of the attacker. Specially, IPSec security associations are utilized to set up authentication tunnels dynamically in order to find out which link is used to carry the attacker's traffic.

4.2 Security Associations with IPSEC/IKE

The key concept in IPSEC[22] is the **security association (SA)** relation between two network entities. The basic service options on a particular SA include authentication and encryption, while Internet Key

Exchange (IKE) [14] is designed for SA establishment, negotiation, and tear-down. The network administrators or designers need to specify the policy and decide *where to set up a particular SA*. An important observation is that it depends on where the SAs have been established to decide the amount of source identification information being provided by examining the attack packet's IPSEC header. For example, if a packet's IP source address is `152.1.75.162` and this packet has been authenticated with IPSEC/AH (transport mode) from `152.1.75.162`, then we are sure that the source IP address is trusted. On the other hand, if the same packet is authenticated with IPSEC/AH (tunnel mode) from `152.1.75.129`, then we can not be sure that this packet is really from `152.1.75.162`. However, we do know that this packet must have been forwarded by `152.1.75.129`. If this is an attack, then we need to further investigate this attack from `152.1.75.129` and beyond. Similarly, if a packet has not been authenticated by `152.1.75.129`, then we also know that it has not gone through that router's IPSec module.

Certainly, one obvious way to ensure trusted IP source addresses is to establish IPSEC security associations (transport mode probably) between any two network entities that might need to communicate with each other. This approach is too expensive and static. If we use IPSEC everywhere, then we need to pay for the IPSEC processing overhead even when there are no attacks. Ideally, we would like to establish SAs in a more flexible way. This leads to the concept of **dynamic security associations**: *in order to efficiently identify the attack sources, the DECIDUOUS module will* **dynamically decide** *where and when to establish IPSEC SAs.*

A network entity supporting IPSEC/IKE consists of three conceptual modules as shown in Figure 4: IKEd (IKE daemon), SAdB (Security Association database), and SPD (Security Policy Database). The entries in the Security Policy database are maintained either manually or by a security management agent (*e.g.,* SMA in the Celestial project [13] or BBN's Policy-based Security Management System (PSMS) project [35]. Each SPD entry normally has a "selector" pointing to a "bundle" of SAs defined in the SAdB. When a particular SPD entry's selector is NULL, the IKEd will be triggered to establish a bundle of SAs. The local IKEd will first negotiate with a remote IKEd for security options and secret key exchange. The IKE daemon will then update its local SAdB. At this point, the SA establishment process has been completed. To summarize, a dynamic SA is established by dynamically updating the Security Policy database.

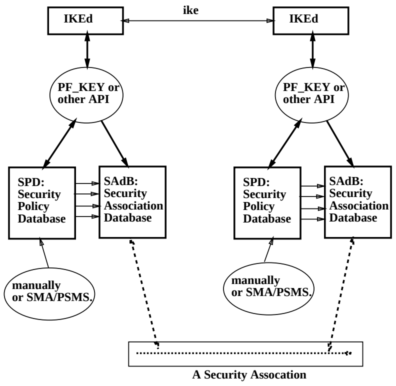

Figure 4: IPSEC Architecture

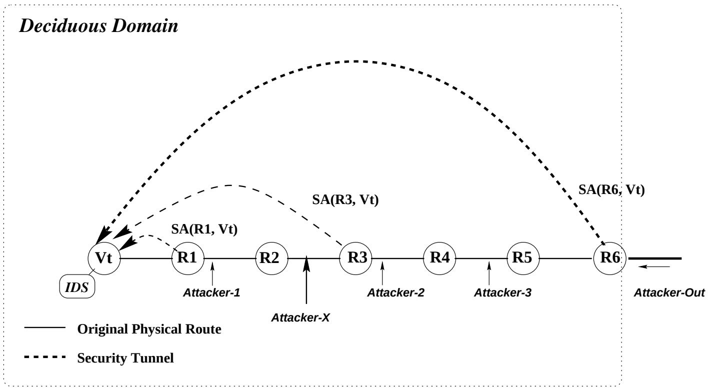

Figure 5: DECIDUOUS Concept in an autonomous domain

5 DECIDUOUS: Dynamic Authentication Tunnel Deployment

The premise of the DECIDUOUS approach is that *if an attack packet has been correctly authenticated by a certain router, that attack packet must have transited that router.* Authentication information here can partition the possible packet origins into two disjoint areas: one with authentication and one without.

Let's again consider the seven-node network under a single administrative domain (i.e., AS, Autonomous System) as shown in Figure 5. We assume that IPSec security associations (SA) can be built among these seven nodes. Let node v_t represents the victim node and $SA(v_s, v_d)$ (dashed line) represents a *Security Association (IPSec/AH tunnel)* from v_s to v_d. This implies that router v_s (if it is good) will forward and authenticate (with tunnel mode) all the packets destined to target v_d. When the next attack is detected by IDS, we need to decide whether the attack packet itself has been authenticated by v_s or not.

Now IDS has sensed that an attack is ongoing. The question is where the attack is from. First, DECIDUOUS will choose the trusted boundary router R_6 to establish an SA (IPSec/AH tunnel) between R_6 and target V_t. DECIDUOUS then determines, by the following attack packets, whether it is an intra-domain attack or an inter-domain attack. If IDS detect a packet with $SA(R_6, V_t)$ in its authentication header list, which indicates that it has been authenticated properly by router R_6, then DECIDUOUS can conclude that the attack is coming from outside this autonomous domain and will launch the collaboration protocol described later to collaborate other DECIDUOUSs in different domains to do inter-domain tracing. Otherwise, the attack is from this domain and DECIDUOUS will systematically deploy different SAs to get more information about the attack path and find the attacker within certain range of area. In short, a simplified DECIDUOUS main algorithm can be stated as in Algorithm 1.

Algorithm 1 DecidMain()

while attack packet pkt_i coming and attackAlarmOn == true **do**
  if checkIntraDomain(pkt_i) == true **then**
    intraDomainTrace(pkt_i); // intra-domain search
  else
    interDomainTrace(pkt_i); // inter-domain search
  end if
end while

5.1 Intra-Domain Tracing

If the attacker is from the intra-domain and DECIDUOUS can do a recursive binary search in this domain until it can conclude the attacker's location within a certain range. In the example of Figure 5, DECIDUOUS will choose the middle point R_3 to establish an SA $SA(R_3, V_t)$ between R_3 and V_t. If the following attack packets are authenticated by $SA(R_3, V_t)$ but not $SA(R_6, V_t)$, the attack is between node 3 and 6, which can be represented by a trace attack zone $AZone[3, 6)$. If it has not been authenticated properly by R_3, then we can conclude that at least one attack source is from either R_3, R_2, R_1, V_t, or the 3 links among these 4 nodes, i.e., $AZone[0, 3)$. Within $AZone[0, 3)$, DECIDUOUS then chooses another middle point R_1 to establish an SA, $SA(R_1, V_t)$, to V_t.

DECIDUOUS has now set up two security associations, $SA(R_1, V_t)$ and $SA(R_3, V_t)$ in the domain. Later on, if IDS receives packets without any SA in the packet authentication list header, attacker is within the area between node R_1 and node V_t. If IDS receives packets with $SA(R_1, V_t)$ but without $SA(R_3, V_t)$, DECIDUOUS will conclude that attacker's location is between node 1 and 3 (i.e., $AZone[1, 3)$). Suppose the attacker is within $AZone[1, 3)$, and then DECIDUOUS will trace this area by establishing another SA from R_2 to node V_t. As shown in Figure 5, if attacker is $Attacker_X$, next time IDS will receive attack packets with $SA(R_2, V_t)$ but without $SA(R_3, V_t)$. After that, DECIDUOUS can conclude that the attacker's location is between node R_2 and R_3 inclusively. This kind of tracing process can be generalized to handle *any network topology* with multiple attackers by introducing the concepts of *Protection Cut*, *CutSet* and *AZone* described in the following sections. We present an algorithm to convert the general topology into the linear topology we need. When we deal with a victim of a general graph topology, we must first find a **CutSet** on the graph.

5.2 General Topology and Linear Topology

Definition 5.1 (Shortest Distance Hop, $SDH(v_x, v_y)$) *Let $G = (V, E)$ be an undirected graph and v_x and v_y are two distinct vertices. The shortest distance hop between v_x and v_y is the number of hops on the shortest path between v_x and v_y.*

Definition 5.2 (Protection Cut $pCut^d_{v_t}$ and CutSet $CutSet_{v_t}$) *Let $G = (V, E)$ be an undirected graph and v_t be the target node to be protected. A d-distance protection cut for v_t is,*

$$pCut^d_{v_t} = \{v_x \mid (v_x \in V) \wedge (SDH(v_x, v_t) = d)\}$$

That is, given a specific node v_t, $pCut^d_{v_t}$ is the set which contains all the nodes that have a same exactly d hops to the v_t. Sometimes, d is called the index of a protection cut.

The CutSet of v_t is an ordered set of the protection cuts of v_t node, such that

$$CutSet_{v_t} = \{pCut^i_{v_t} \mid 0 \le i \le n-1, i \in \mathcal{N}\}$$

where $n = |CutSet_{v_t}|$ *the number of the protection cuts of* v_t. *Sometimes, we use* $CutSet_{v_t}[i]$ *to represent* $pCut^i_{v_t}$ *which contains all the nodes with* i *hops away from* v_t.

In other words, the nodes with the same number of hops from node v_t form a *Protection Cut*. The **CutSet** of node v_t is the set of these protection cuts.

Algorithm 2 CutSet Linear Topology Translation: CutSetTrans($Node_{target}$)

Node $v_t = Node_{target}$; // indicate the target root, first unexplored vertex
Vector $Q_{totalNode}$; // Initially contains all the network nodes
Vector Q_{pCut} = new Vector(); // contains nodes in a specific Cut
Vector Q_{CutSet} = new Vector(); // contains all the CutSets
int $N \leftarrow |Q_{totalNode}|$; // total number of node
int $curHops \leftarrow 0$; // the number of hops from v_t
int $Cnt_{node} \leftarrow 0$; // Count how many node have been explored
$v_t.cutId \leftarrow curHops$
Q_{pCut}.add(v_t); // Initially only one $CutSet_0$ which contains the target node v_t
$Cnt_{node} \leftarrow Cnt_{node} + 1$;
Q_{CutSet}.add(Q_{pCut}); // Add a protection cut to the CutSet
while $Cnt_{node} < N$ **do**
  Vector $Q_{tmp} = Q_{CutSet}$.last(); // pick the newest cut from the last element of the CutSet
  Q_{pCut} = new Vector(); // Create a new protection cut
  $curHops \leftarrow curHops + 1$
  for all Node $v \in Q_{tmp}$ and ($Cnt_{node} < N$) **do**
    for all Node n_i adjacent from v and ($Cnt_{node} < N$) **do**
      if $n_i.cutId == -1$ **then**
        $n_i.cutId \leftarrow curHops$
        Q_{pCut}.add(n_i);
        $Cnt_{node} \leftarrow Cnt_{node} + 1$;
      end if
    end for
  end for
  Q_{CutSet}.add(Q_{pCut}); // Add a protection cut to the CutSet
end while
print("Done: Q_{CutSet} contains all the protection cuts");

Suppose we have a graph $G =< V, E >$ represented by a data structure of adjacency lists, in which each vertex keep a list of its neighbors. The translation from general topology to linear *CutSet* topology can be done like a Breadth-First Search algorithm shown as in the Algorithm 2. Every node is associated with a cut index, *cutId*, which indicates if the node has already been counted in some $pCut^{cutId}$ or not. Initially, all the *cutId* is set to -1. In the algorithm, *Vector* represents a special data structure like a dynamic array or container with arbitrary size (runtime allocated), and it can contain any class of other data objects. The statement $Cnt_{node} < N$ is the criterion to stop the algorithm when all the nodes have been explored. Note that again $CutSet_{v_t}[i]$ is a set of nodes which are in the same protection cut, $pCut^i_{v_t}$, with a shortest distance hop i away from v_t. Each node has its own CutSet topology which is different from the others.

5.3 An Example of CutSet Translation Algorithm

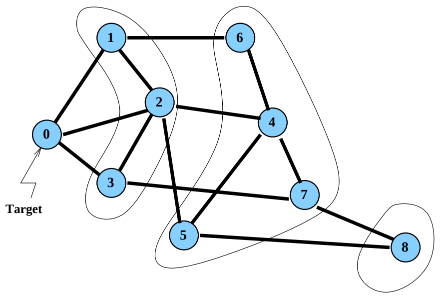

Figure 6: Protection Cuts in a General Network Topology

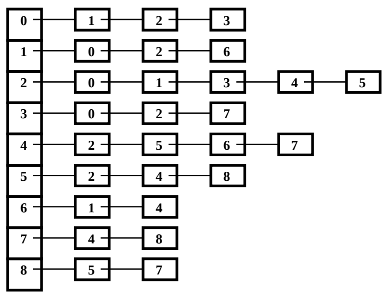

Figure 7: Adjacency List representation of a general network topology

An example of *CutSet* concept is illustrated in Figure 6 and Figure 8. Assume node 0 is the target node to be protected. Figure 7 shows an adjacency list presentation for this network topology.

1. Initially, only one cut formed in CutSet $\mathcal{C}$. That is,

   $pCut^0 = \{0\} = \mathcal{C}[0] \quad \Longrightarrow \quad \mathcal{C} = \{\{0\}\},\ Cnt_{node} = 1$

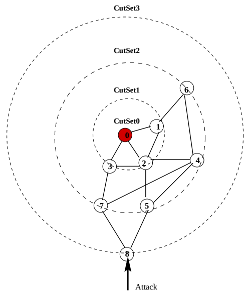

Figure 8: Protection CutSet concept

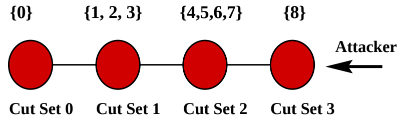

Figure 9: CutSet linear topology translation example

2. Investigate the adjacency list of node 0, we get
   $pCut^1 = \{1,2,3\} = \mathcal{C}[1] \quad \Longrightarrow \quad \mathcal{C} = \{\{0\},\{1,2,3\}\},\ Cnt_{node} = 4$
3. Investigating the adjacency lists of node 1, 2, and 3, we get
   $pCut^2 = \{6,4,7,5\} = \mathcal{C}[2] \quad \Longrightarrow \quad \mathcal{C} = \{\{0\},\{1,2,3\},\{6,4,7,5\}\},\ Cnt_{node} = 8$
4. Investigating the adjacency lists of node 6, 4, 7 and 5, then
   $pCut^3 = \{8\} = \mathcal{C}[3] \quad \Longrightarrow \quad \mathcal{C} = \{\{0\},\{1,2,3\},\{6,4,7,5\},\{8\}\},\ Cnt_{node} = 9$
5. Since all the nodes have been counted (by checking $Cnt_{node} == 9$), algorithm will stop here. Finally,
   $\mathcal{C} = \{\{0\},\{1,2,3\},\{6,4,7,5\},\{8\}\}$

The CutSet linear topology will be like $\{0\}$ – $\{1, 2, 3\}$ — $\{4, 5, 6, 7\}$ — $\{8\}$ shown in Figure 9. Any path to node 0 should go through in the order of protection cuts of this CutSet. That is, the traffic from node 8 to node 0 must visit at least one node of $\{4, 5, 6, 7\}$, and then at least one node of $\{1, 2, 3\}$.

Definition 5.3 (Authentication Verification Function for CutSet) *$pCut^d$ authentication is* **enabled** *when all of its nodes enable their authentication functions. An authencation verification function for an enabled $pCut^d$ can be defined as,*

$$authCutVer(pkt, pCut^d) = \begin{cases} true & \text{if } \exists v_i \in pCut^d \text{ such that } authVer(pkt, v_i) = true \\ false & \text{if } \forall v_i \in pCut^d,\ authVer(pkt, v_i) = false \end{cases}$$

where $authVer(pkt, v_i)$ is a function to verify authentication node. Note that $authCutVer(pkt, pCut^d) = null$, if $pCut^d$ authentication is **disabled**

In other words, an given authenticated packet *pkt* must transit an enabled protection cut $pCut^d$, if $authCutVer(pkt, pCut^d) == true$. That is, *pkt* must go through *at least one* authentication node of this cut.

Definition 5.4 (Suspect Attacker Zone $AZone$) *Let $\mathcal{C}_{v_t}$ is the protection CutSet of v_t formed by the algorithm 2. Each suspect attacker zone $AZone \subseteq \mathcal{C}_{v_t}$ can be represent as a range $[pCut^{d_l}, pCut^{d_u})$, such that $pCut^{d_l}$ and $pCut^{d_u}$ are two protection cuts, and*

$$d_l = max\{i|\ pCut^i \in \chi\},$$

$$d_u = min\{j|\ pCut^j \in \zeta\}$$

where $\chi = \{\ pCut^i \mid authCutVer(pkt, pCut^i) = true,\ pCut^i \in \mathcal{C}_{v_t}\}$ contains cuts which pkt has gone through , and $\zeta = \{\ pCut^j \mid authCutVer(pkt, pCut^j) = false,\ pCut^j \in \mathcal{C}_{v_t}\}$ contains cuts which pkt has not visited.

Note that $\chi \cup \zeta$ may not be equal to $\mathcal{C}$ because authentication tunnel SAs are "created-on-demand", there may be some $pCut^e$ which have not been enabled, thereby $authCutVer(pkt, pCut^e) = null$. Furthermore, given attack packets from a single attacker, if AZone$[pCut^{d_l}, pCut^{d_u})$ covers the attacker, then

$$authCutVer(pkt_i, pCut^{d_l}) = true \text{ and } authCutVer(pkt_i, pCut^{d_u}) = false\ ,$$

where d_l is called the minimum cut index, and d_u is the maximum cut index of this AZone. Each AZone will keep shrinking its size in the tracing algorithm until the suspect area becomes a reasonable small size. That is, d_l and d_u will be changed dynamically within the tracing period.

Definition 5.5 (Function $chkAttInZone(pkt_{att}, AZone)$**)** *Given an attack packet* pkt_{att}*,*

$$chkAttInZone(pkt_{att}, AZone) = \begin{cases} 0 & \text{if } authCutVer(pkt_i, pCut^{d_l}) = false \text{ and } authCutVer(pkt_i, AZone.pCut^{d_u}) = false. \\ 1 & \text{if } authCutVer(pkt_i, AZone.pCut^{d_l}) = true \text{ and } authCutVer(pkt_i, AZone.pCut^{d_u}) = false. \\ 2 & \text{if } authCutVer(pkt_i, AZone.pCut^{d_u}) = true. \end{cases}$$

which verifies where the attacker's location is related to AZone partitions.

The whole linear topology within one domain is partitioned by several non-overlap AZones and some uncovered areas. That is, intra-domain attackers can be identified by several disjoint attacker zones such that $AZone_i \cap AZone_j = \phi, \forall\, i \neq j$, which contains all suspicious area of attacker's location. Algorithm 3 shows how to use one general strategy to trace attackers and dynamically create a disjoint AZone for a new suspect area within a DECIDUOUS domain. Different strategies for different types of attacks can be used for optimization.

Algorithm 3 intraDomainTrace(pkt_{att}): tracing algorithm with a general strategy

int $\epsilon \leftarrow 1$;
int $d_l = 0$; // initial for new AZone.MinCutInd
int $d_u = CutSet.size()$; // initial for new AZone.MaxCutInd
Boolean $New_{need} \leftarrow true$;
Vector $AZoneSet$; // contains all the AZones
IDSINFO $info_{IDS}$ = IDSInfo(pkt_{att});
for all $AZone_i \in AZoneSet$ and $New_{need} = true$ **do**
  int $Ret \leftarrow$ **chkAttInZone**$^*(info_{IDS}, AZone_i)$;
  if Ret == 1 **then** {identify attacker is in this AZone}
    $New_{need} \leftarrow false$; // Don't have to check the other AZones and stop the loop
    if $AZone_i.attackerFound == false$ **then** {attacker found in this AZone}
      $AZone_i$.**BinarySearch**($info_{IDS}$); // reduce the range
    else
      print("Attacker found within $AZone_i.d_l$ and $AZone_i.d_u$ area");
    end if
  else if Ret == 2 **then** {attacker is beyond MaxCutInd of this AZone}
    $New_{need} \leftarrow true$;
    if $AZone_i.d_u < d_u$ **then**
      $d_l \leftarrow max(d_l, AZone_i.d_u)$;
    end if
  else {Ret == 0; attacker is within MinCutInd of this Zone}
    $New_{need} \leftarrow true$;
    if $d_l < AZone_i.d_l$ **then**
      $d_u \leftarrow min(d_u, AZone_i.d_l)$;
    end if
  end if
end for
if New_{need} == true **then** {create a new AZone if necessary.}
  $AZoneSet.add(new\ AZone(d_l, d_u))$;
end if

In this strategy, we assume the v_t itself is trusted so that $authCutVer(pkt, pCut^0)$ is always *true*. Note that $pCut^{|\mathcal{C}|} = \phi$ due to the index d of $pCut^d$ starting from 0, so $authCutVer(pkt, pCut^{|\mathcal{C}|})$ is defined to be always *false*. Initially the $AZone_0$ must cover the whole topology so $AZone_0 = [pCut^0, pCut^{|\mathcal{C}|})$. *IDSInfo* is a IDS function to retrieve the authentication information associated with each attack packet. In practice, because IDS only sends *IDSINFO* messages to DECIDUOUS instead of the whole packet, the function authentication verification should be like $authVer^*(IDSinfo, v_i)$ and $authCutVer^*(IDSinfo, pCut^d)$.

The algorithm will first try to find which AZone the attacker belongs to, and then will do a binary search on that AZone as shown in Algorithm 4, trying to shrink the AZone area. If there is no AZone covering this attacker, it will create another new AZone with a proper range area based on the information inferred. Therefore, even multiple attackers in different areas can be caught by different AZones. The stop criterion for BinarySearch is $\epsilon = 1$ which means that this search keep going, and AZone will shrink until $|d_u - d_l| <= \epsilon$. The Function **SetSA**($pCut^d$) will set up security association rules in every node $n_i \in pCut^d$, and the opposite **TeardownSA**($pCut^d$) will teardwon the corresponding SAs in all the nodes of $pCut^d$.

5.4 An Example of Multiple Intra-domain Attackers

Consider an example again in Figure 5. Three attackers are sending attack packets toward the victim v_t in turns. DECIDUOUS has established $SA(R_6, v_t)$ and determines that the attack is from intra-domain by packets received.

Algorithm 4 BinarySearch($info_{IDS}$): search algorithm in AZone for a given attack packet

```
int ϵ ← 1;
int d_tdown ← −1; // keep the teardown Cut
if pCut^{d_mid} ≠ −1 then {Initial d_mid == −1 and no SA in pCut^{d_mid}}
  if authVer*(info_IDS, pCut^{d_mid}) == true then {Attacker check-in MidCutInd}
    d_tdown ← d_l;
    d_l ← d_mid;
  else if authVer*(info_IDS, pCut^{d_mid}) == false then {Attacker not check-in MidCutInd}
    d_tdown ← d_u;
    d_u ← d_mid;
  else {authVer*(info_IDS, pCut^{d_mid}) == null}
    print("Initial d_mid == −1");
  end if
end if
// check if attacker has been found in a certain range
if |d_u − d_l| ≤ ϵ then
  print("Attacker found within MinCutInd d_l and MaxCutInd d_u");
else {Shrinking area and keep tracing}
  d_mid = (d_l + d_u)/2;
  SetSA(pCut^{d_mid}); // Establish SAs for the middle Cut
end if
if d_tdown ≠ −1 then {Teardown SA in pCut^{d_tdown}}
  TeardownSA(pCut^{d_tdown});
end if
```

1. Receiving an attack packet pkt_0 (assuming from an unknown attacker-1). DECIDUOUS forms CutSet $\mathcal{C}_{v_t}$ and $AZone_0$ for v_t, $\mathcal{C}_{v_t} = \{\{0\},\{1\},\{2\},\{3\},\{4\},\{5\},\{6\}\}$, where $|\mathcal{C}_{v_t}| = 7$. Thus, $AZone_0 = [pCut^0, pCut^7) = [\{0\}, \phi)$, $AZoneSet = \{AZone_0\}$. $\Longrightarrow$ DECIDUOUS $SetSA(pCut^3)$.

2. Receiving pkt_1 (from attacker-2), $chkAttInZone(pkt_1, AZone_0) = 1$, and $authCutVer(pkt_1, pCut^3) = true$. Thus, $AZone_0 = [pCut^3, pCut^7) = [\{3\}, \phi)$. $\Longrightarrow$ DECIDUOUS $SetSA(pCut^5)$, and $TeardownSA(pCut^0)$ which has no effect.

3. Receiving pkt_2 (from attacker-3), $chkAttInZone(pkt_2, AZone_0) = 1$ and $authCutVer(pkt_2, pCut^5) = false$. Thus, $AZone_0 = [pCut^3, pCut^5) = [\{3\}, \{5\})$. $\Longrightarrow$ DECIDUOUS $SetSA(pCut^4)$, and $TeardownSA(pCut^7)$ which has no effect.

4. Receiving pkt_3 (from attacker-1), $chkAttInZone(pkt_3, AZone_0) = 0$, so a new AZone is created, $AZone_1 = [pCut^0, pCut^3) = [\{0\}, \{3\})$, $AZoneSet = \{AZone_0, AZone_1\}$. $\Longrightarrow$ DECIDUOUS $SetSA(pCut^1)$.

5. Receiving pkt_4 (from attacker-2), $chkAttInZone(pkt_4, AZone_0) = 1$, and $authCutVer(pkt_4, pCut^4) = false$. Thus, $AZone_0 = [pCut^3, pCut^4) = [\{3\}, \{4\})$. Because $|4-3| \le \epsilon$, attacker-2 is found within the area between $pCut^3$ and $pCut^4$, inclusively. $\Longrightarrow$ DECIDUOUS $TeardownSA(pCut^5)$.

6. Receiving pkt_5 (from attacker-3), $chkAttInZone(pkt_5, AZone_0) = 2$, and $chkAttInZone(pkt_5, AZone_1) = 2$, so a new AZone is created, $AZone_2 = [pCut^4, pCut^7) = [\{4\}, \phi)$, $AZoneSet = \{AZone_0, AZone_1, AZone_2\}$. $\Longrightarrow$ DECIDUOUS $SetSA(pCut^5)$.

7. Receiving pkt_6 (from attacker-1), $chkAttInZone(pkt_6, AZone_1) = 1$, and $authCutVer(pkt_6, pCut^1) = true$. Thus, $AZone_1 = [pCut^1, pCut^3) = [\{1\}, \{3\})$. $\Longrightarrow$ DECIDUOUS $SetSA(pCut^2)$.

8. Receiving pkt_7 (from attacker-2), $chkAttInZone(pkt_7, AZone_0) = 1$, and $AZone_0.attackerFound = true$, so it has been covered.

9. Receiving pkt_8 (from attacker-3), $chkAttInZone(pkt_8, AZone_2) = 1$, and $chkAttInZone(pkt_5, pCut^5) = false$, Thus, $AZone_2 = [pCut^4, pCut^5) = [\{4\}, \{5\})$. Because $|5-4| \le \epsilon$, attacker-3 is found within the area between $pCut^4$ and $pCut^5$, inclusively.

10. Receiving pkt_9 (from attacker-1), $chkAttInZone(pkt_9, AZone_1) = 1$, and $authCutVer(pkt_9, pCut^2) = false$. Thus, $AZone_1 = [pCut^1, pCut^2) = [\{1\}, \{2\})$. Because $|2-1| \le \epsilon$, attacker-1 is found within the area between $pCut^1$ and $pCut^2$, inclusively. $\Longrightarrow$ DECIDUOUS $TeardownSA(pCut^0)$ which has no effect.

11. After this point, any packet from these three attackers will be covered by one of three AZones and attacks were caught.

As the example shown, in a topology with a CutSet of size 7, it requires only about 10 critical packets to catch all three attackers in different areas. In theory, it requires only $O(lg\ d)$ critical packet rounds to catch a single attacker in a CutSet of size d. In practice, the factors affect the tracing time and the actual number of packets needed include the speed of SA establishment, how many attackers, attackers' locations and the attack schemes.

6 Inter-Domain Tracing by a Collaboration Protocol

As evident from the above, the main requirement for this approach is the ability to establish IPSec security association with routers/gateways and to be able to correctly correlate the attack packets with the security associations they traveled on. While it may be generally possible for a DECIDUOUS daemon to establish SAs with routers within its domain, this may not be the case with routers in the other administrative domains. Consequently, there is a need for two or more DECIDUOUS daemons to collaborate with one another to further trace the origin of an attack. This section describes a protocol to provide the communication and collaboration between two DECIDUOUS daemons for the purpose of attack source identification.

Generally, it may not be possible for the local DECIDUOUS($\mathcal{D}_{local}$) to establish SAs with interior routers in a remote domain. This may be because:

1. The border gateway of a domain may have a policy disallowing any encrypted traffic through it.
2. The identity of the interior routers may not be obtainable due to administrative policy issues.
3. The local/remote domains may be using private IP addresses internally and using Network Address Translator (NAT) at the boundary to perform address translation.

As mentioned before, if the attack packets have been authenticated by one or more border routers, the attack may be originating from outside of this domain. While it is not possible to establish SAs directly with the remote domain routers, the local DECIDUOUS daemon $\mathcal{D}_{local}$ will attempt to initiate a collaboration session with the remote DECIDUOUS $\mathcal{D}_{remote}$ of the remote domain. The remote DECIDUOUS will function as a proxy for the local DECIDUOUS and will manage the process of further identifying the attack source within its domain. Once the border router(s) in the local domain have been identified, attack tracing can continue by collaborating with the domains that are adjacent to the border router(s) through a special collaboration engine (CE) module of DECIDUOUS. This approach can be used to iteratively trace down the attackers across several domains. Note that if the attack packets transit multiple local border routers, multiple collaboration sessions may need to be established if it appears that the attack(s) may be originating in multiple domains.

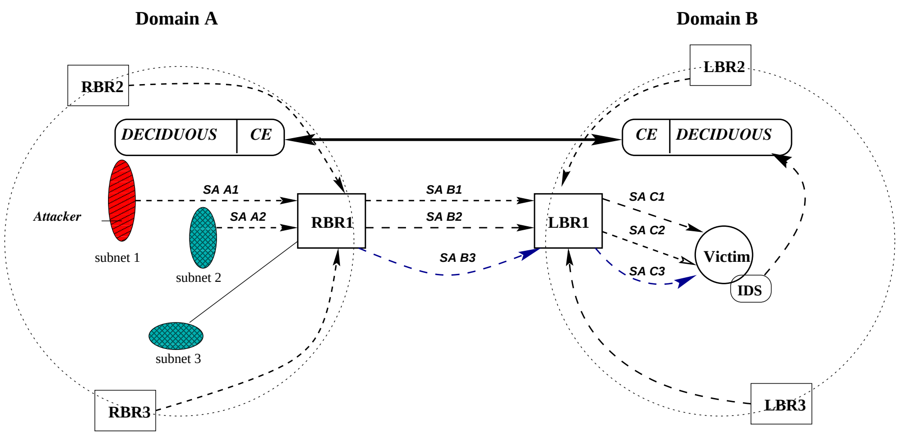

Figure 10: A collaboration example

Figure 10 shows a simplified example of the collaboration protocol. Assume that the DECIDUOUS daemon of the domain B, $\mathcal{D}_B$, has identified that an attack stream has been authenticated by the local border router LBR_1 which is connected to a remote border router RBR_1 of the domain A. The local DECIDUOUS $\mathcal{D}_B$ then initiates a collaboration session with the remote DECIDUOUS $\mathcal{D}_A$ through the communication of CE_B and CE_A. When the collaboration session is initiated, the remote CE_A will establish n (n=3 in figure above, this number is negotiated between the two collaboration engines) security associations $SA(RBR_1, LBR1)$, SA_{B1}, SA_{B2}, and SA_{B3}, each with a different Security Parameter Index (SPI) value, from router RBR_1 to router LBR_1. CE_A will then communicate the SPI values to CE_B. CE_B will also establish an equal number of security associations $SA(LBR_1, Victim)$, SA_{C1}, SA_{C2}, and SA_{C3} from LBR_1 to the victim node or the router serving the victim's subnet. CE_B will also install a mapping table in LBR_1 such that there is one-to-one tunnel mapping from $\{SA_{B1}, SA_{B2}, SA_{B3}\}$ to

$\{SA_{C1}, SA_{C2}, SA_{C3}\}$ based on SA information and their associated SPI values. For example, a simple mapping can be setup so that packets coming in on SA_{B2} will be forwarded on SA_{C2}. More complex mappings can also be established, if necessary.

This forwarding technique based on SA value has not yet been supported by current IPSec. We therefore implemented what we refer to as *PHIL-Switching* with which *an IPSec node is capable of forwarding packets based on the incoming SA SPI values and other security parameters.* What this accomplishes, in effect, is a form of authenticated virtual circuit switching between the attack source and the victim that can be used to accurately trace the attacker's location. As a result, by observing the SA a packet traverses in domain B, it is possible to deduce the SA it must have traversed in domain A.

At the same time, $\mathcal{D}_A$ will begin to trace the attack within domain A, using the techniques outlined above for intra-domain attack isolation. As far as $\mathcal{D}_A$ is concerned, it is simply tracing an intra-domain attack. All of the collaboration issues are handled by the two collaboration engines in a manner transparent to $\mathcal{D}_A$.

Note that the primary purpose of the collaboration is to be able to correctly identify the attack origin(s) based on the observations made in the local domain. Intrusion information from the local IDS will be shared with the remote DECIDUOUS agent, and remote DECIDUOUS agent will decide where to set up additional SAs within its domain. That is, when $\mathcal{D}_B$ receives IDSinfo from IDS in domain B, CE_B and CE_A must translate this information to what the remote $\mathcal{D}_A$ can process. For example, when $\mathcal{D}_B$ receives IDSinfo containing SA_{C1}, CE_B will send IDSinfo with SA_{B1} by PHIL-Switching table-lookup in LBR_1. Similarly, when CE_A receives IDSinfo from CE_B, containing SA_{B1} information, it will translate it by PHIL-Switching table-lookup in RBR_1 and send IDSinfo with the SA_{A1} to $\mathcal{D}_A$. In other words, CE_A here will act as the IDS in the intra-domain algorithm to send correct IDSinfo to $\mathcal{D}_A$.

In this example, $\mathcal{D}_A$ decided to set up SA_{A1} from subnet 1 to RBR_1 and SA_{A2} from subnet 2 to RBR_1. CE_A will configure the PHIL-Switching table to map $SA_{A1} \rightarrow SA_{B1}$ and $SA_{A2} \rightarrow SA_{B2}$. That is, policies at RBR_1 state that packets coming in on SA_{A1} will be forwarded to SA_{B1}, and packets coming in on SA_{A2} will be forwarded to SA_{B2}. Policies at LBR_1 state that packets coming in on SA_{B1} will be forwarded to SA_{C1} and packets coming in on SA_{B2} will be forwarded to SA_{C2}. By dynamically setting up SAs and the use of PHIL-Switching, DECIDUOUS can trace the packets' origin across many domains regardless of spoofed IP addresses.

7 Implementation, Testbed and Experimental Results

DECIDUOUS utilizes the IPSEC header information to locate the attack sources. In the previous isolation examples, we inherently made **an important assumption**: *the IDS module, when detecting an attack, can access the IP authenticated header of the packet(s) that carried this attack.* For a network layer IDS (*e.g.,* the JiNao network infrastructure intrusion detection system [17, 18]), usually the information is immediately available. However, if the attack can only be detected in the application layer, then the IDS is normally unable to identify the original IPSEC AH-tunnel header. We therefore developed a new socket interface API by which the applications like IDS or SNMP daemon can retrieve a *Packet Header Identification List (PHIL)* [26, 4] containing all the IP-layer authentication information[1] related to this particular attack packet from the kernel. Alternately, this application can pass the PHIL identifier to another process for further attack handling. Note that in order to isolate application level attacks aimed at an end host, the end host must be IPSEC capable and has the PHIL-capable IDS module running locally.

Figure 11 shows the architectural diagram of a target host protected by the DECIDUOUS daemon (DECId) process. The DECId talks to one or more IDS components to collect intrusion information. After obtaining authentication information from the IDS modules, the DECId will decide whether the attack is in one of the already-established attacker zones. If not, a new zone will be created for tracking a new attacking point. In either case, the DECId will decide whether a new SA needs to be established or an existing SA needs to be torn down. The DECId will update the security policy database (SPD) either directly or through a security management system. The interface between DECId and SPD is currently implementation dependent. We implemented a policy server to handle the communication between DECId and IPSec SPD. Different vendors may provide different configuration API for access to the SPD. Therefore, in order to port DECIDUOUS to another platform, we may need to re-implement the DECId/SPD interface.

[1] Note that with the tunnel mode, an incoming IP packet could have more than one authentication header.

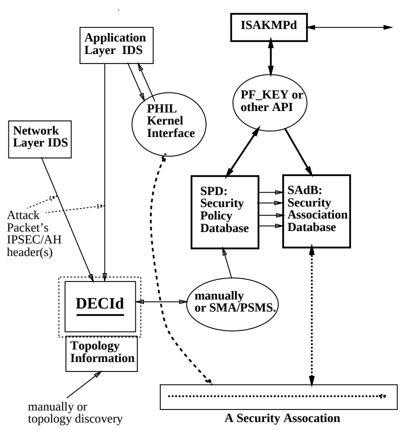

Figure 11: The DECIDUOUS Architecture

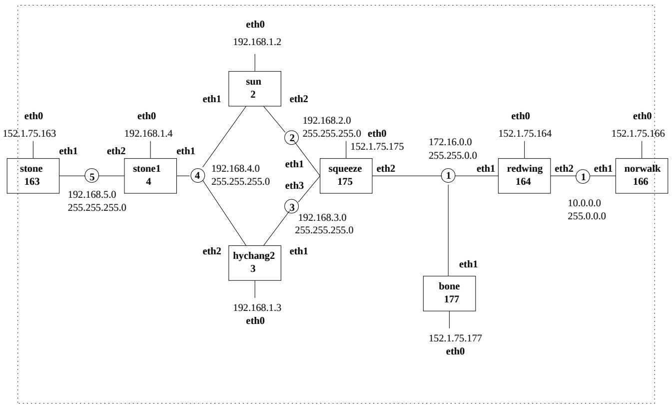

Figure 12: The DECIDUOUS Network Testbed Configuration

Moreover, there is a lot of overhead in setting up a security association, and maintaining its parameters such as keys exchange, SPD and security association database (SAdB) update. Such an overhead is acceptable for most network applications, where only a few SAs are required, and there is network traffic for most of the SAs established. But consider a network application, such as DECIDUOUS, which needs to communicate securely, but does not know in advance the source or destination of the network traffic. To cover all possible sources and destinations which may be of interest, such a network application will need to create a large number of security associations. This will incur unnecessary computational overheads. Especially, if only a few (possibly none) of the established SAs are ever used, *there is a waste of resources in setting up security associations which will never be used.* Hence, we developed what we so called *traffic-driven SA establishment* in *DIANA*, a policy management module [15, 5] for an existing implementation of IPSec called FreeS/WAN. DIANA maintains a SPD, which specifies the policies that determine the processing of all outbound IP traffic. By intercepting outgoing IP packets from the operating system kernel, DIANA matches them with policies specified in the SPD and establishes the SAs if a matching policy is found. That is, the network traffic will thus act as a *trigger* for establishing security associations only when there is a matching found. This will eliminate the overheads associated with the creation of unnecessary SAs.

7.1 Testbed and Experimental Results

The DECIDUOUS system has been implemented in C/C++/Java on top of Linux 2.0.36 with FreeS/WAN 1.0 support. The kernel has been enhanced to support: divert sockets, PHIL API and PHIL-Switching, and DIANA (dynamic and traffic driven SAs). The system has been tested and demonstrated on a testbed network consisting of 8 Linux PC routers as shown in Figure 12. We also built a network-based IDS that will utilize the PHIL-API to obtain the IPSec header information about the detected attack packets. The packet header along with the PHIL information will be sent to the DECIDUOUS daemon for locating the true sources. Any machine here can be the victim as well as the attacker. As an example presents here, we let the machine *norwalk* to be the victim with PHIL-capable IDS. DECIDUOUS sits at the machine *bone*. Every router is IPSec capable and is installed with the policy server DIANA. We have performed various experiments on our testbed:

Simple Single Source: In this experiment, we randomly choose one router as the attack source launching spoofed attacks. By the strategy described in Section 5.1, all the possible attackers can be traced down to their correct origins.

Simple Multiple Sources: The same as the previous experiment except we have more than one different attack sources. The timing race condition under this case is more severe. Different packets may come from different attackers, interspersed with each other. In order to correctly identify the true sources, we have to wait for about 300 msec for checking all the subsequent packets to safely make the decision of where to set up the next SA

Coordinated Multiple Sources: We still have two sources but only one of them attacks the victim, while the other source attacks the DECIDUOUS control messages by randomly dropping them. This coordinated attack will significant slow down the performance of source identification, but will not affect the correctness of the final results. Also, because we have not implemented the strategy to identify the "dropping sources," under this case, we can only identify the source who attacks the victim.

Category	Attackers	Detection Time	Total # of Pkts
Single Attacker	redwing	25841.7 msec	20
	squeeze	26147.7 msec	20.5
	sun	20089.2 msec	14.6
	stone1	19790.7 msec	14.1
Multiple Attackers	squeeze & redwing	40914.4 msec	70.2
	sun & redwing	37687.7 msec	63.9
	stone1 & redwing	41742 msec	71.4
	3-attacker	43136.4 msec	111.4
	4-attacker	46597.5 msec	154.8

Table 1: Results for DEMO TestBed

Table 1 compares some results of tracing single and multiple attackers in our testbed of network diameter 5. 3-attacker case includes attackers from machine stone1 (st1), squeeze (sq), redwing (re), and 4-attacker includes an additional sun (su). The detection time is for the total time in average to find out the sources of all the possible attackers. Total number of packets counts the attack packets obtained since the first attack packet has been sensed. This number depends on how fast the attack packets can be sent and the network speed. Furthermore, for our general strategy, the detection time may also be affected by the actual number of tracing rounds needed, the speed of SA establishment, how many attackers, attackers' locations, and the attack schemes. In single attacker tracing, it takes only less than 30 seconds to find the attacker. Each AZone needs just about three attack tracing rounds ($O(lg\ 5)$) to lock out the attacker's location. Finally, we try an aggressive strategy (establishing more SAs in advance). And, surprisingly, we can get the number down to about 2 to 7 attack packets before identifying the true source.

	$Attacker_0$ Found				$Attacker_1$ Found				$Attacker_2$ Found				$Attacker_3$ Found			
$Test_i$	$\tau_0\ ms$	κ_0	ι_0	α_0	$\tau_1\ ms$	κ_1	ι_1	α_1	$\tau_2\ ms$	κ_2	ι_2	α_2	$\tau_3\ ms$	κ_3	ι_3	α_3
0	23965	69	3	sq	30057	97	1	su	36502	124	1	st1	43522	146	1	re
3	24106	70	3	sq	30203	98	1	su	42645	149	1	st1	42686	123	1	re
4	24185	71	3	sq	30283	99	1	su	42725	150	1	st1	42768	124	1	re
6	25053	73	3	su	42560	102	1	re	48349	149	1	sq	48975	171	2	st1
7	35543	74	3	st1	42183	117	1	re	42613	143	2	su	49904	176	1	sq
9	24608	72	3	su	42138	101	1	re	47902	148	1	sq	48534	170	2	st1
11	24727	72	3	su	42293	101	1	re	47964	148	1	sq	48587	170	2	st1

Table 2: Partial sample results for 4-attacker case

Table 1 shows that although it takes longer time to detect multiple attackers, there is not that much difference from detecting 2-attacker to 3-attacker, or to 4-attacker. Because after all the AZones are formed, they can trace different attackers in different areas simultaneously. Moreover, in our test samples, DECIDUOUS may not found the attackers in the same sequence because of the interleaved attack packets. Table 2 shows a partial test result from the 4-attacker case. Each row represents a test starting with its test number. DECIDUOUS found the $attacker_i$ within τ_i msec since the trace started, with totally κ_i attack packets arrived, and only observed ι_i packet in the corresponding AZone. For example, in the test 0, DECIDUOUS found attacker in squeeze (sq) first within 24 seconds, and then in sun (su) within 30 seconds, and then in the stone1 (st1) within 36.5 seconds, and finally in redwing within 43.5 seconds. But in test 6, DECIDUOUS found attacker in sun first, and then in redwing, and then in squeeze, and finally in stone1.

8 Summary, Conclusions and Future work

This paper describes the major idea and algorithms of the DECIDUOUS security management framework for tracing intrusion sources. We have demonstrated how to tracing multiple intra-domain attackers and how to use the collaboration protocol to trace inter-domain attackers. DECIDUOUS is designed to work with intrusion detection systems (IDS) and intrusion damage control systems (IDCS). The integration of DECIDUOUS, IDS, and IDCS is a powerful architecture for dealing with various network-based intrusions. In general, the design of the DECIDUOUS system can support the following,

- Multiple attackers and victims: DECIDUOUS can trace attacks which may be launched from multiple attack sources, even across different domains, simultaneously toward the several victims. Each victim owns a trace session to traceback the origins of its own attackers by utilizing appropriate strategies and relevant attack information.
- Multiple IDS systems and attack types: As a large corporation or organization may deploy IDS from more than one vendor, DECIDUOUS should be capable of interfacing with multiple IDSs and process information from all of them, and distinguish between multiple simultaneous attack types to launch optimal tracing strategies based on the attack types.
- Logging and Post-Mortem attack trace analysis: DECIDUOUS has capability to log all the intermediate attack zone information and statistic data. In case of system crashing during the tracing, these logs can provide valuable information for later off-line analysis.

Our experiments show that the DECIDUOUS system is reasonably efficient, flexible in supporting different tracing strategies, robust against certain attacks/failures directly on DECIDUOUS itself, and correct in determining the true sources even under more delicate coordinated attacks. Although the algorithms can be optimized further for better performance, the system described here could serve as the basis for future research on different tracing strategies for different type of attacks, and hopefully we can build a DECIDUOUS network on top of the Internet to perform large-scale experiments against real intrusions.

References

[1] A. Beimel and Matt Franklin. Reliable communication over partially authenticated networks. In *Proc. 11th International Workshop on Distributed Algoritms (WDAG)*, 1997. http://www.parc.xerox.com/csl/members/franklin/.

[2] Bellovin. Icmp traceback messages. Internet Draft, March 2000. http://www.ietf.org/internet-drafts/draft-bellovin-itrace-00.txt.

[3] SANS Global Incident Analysis Center. Egress filtering. http://www.sans.org/y2k/egress.htm, February 2000.

[4] H.Y. Chang, S.F. Wu, and et al. Deciduous: Decentralized source identification for network-based intrusions. In *6th IFIP/IEEE International Symposium on Integrated Network Management*. IEEE Communication Society press, May 1999.

[5] H.Y. Chang, S.F. Wu, and et al. Design and implementation of a real-time decentralized source identification system for untrusted ip packets. In *DARPA Information Survivability Conference and Exposition(DISCEX 2000)*. IEEE Computer Society press, January 2000.

[6] H.Y. Chang, S.F. Wu, and Y.F. Jou. Real-time protocol analysis for detecting link-state routing protocol attacks. *submitted to ACM Transaction on Information and System Security for review*, 1999.

[7] Inc. CISCO System. Characterizing and tracing packet floods using cisco routers. http://www.cisco.com/warp/public/707/22.html, 1999.

[8] CNN.com. Cyber-attacks batter web heavyweights strikes on ebay, amazon, cnn.com follow monday yahoo! attack, February 2000. http://www.cnn.com/2000/TECH/computing/02/09/cyber.attacks.01/index.html.

[9] Tom Dunigan. Back tracking spoofed packets. http://www.epm.ornl.gov/ dunigan/oci/bktrk.html, March 2000.

[10] Tom Dunigan. Isp spoof tester. http://www.epm.ornl.gov/ dunigan/oci/spoof.html, April 2000.

[11] P. Ferguson and D. Senie. Rfc 2267: Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing, January 1998. ftp://ftp.isi.edu/in-notes/rfc2267.txt.

[12] V. Gligor. A note on the denial-of-service problem. In *Proceedings of the 1983 IEEE Symposium on Security and Privacy*, pages 139–149. IEEE Computer Society Press, 1983.

[13] F. Gong. Celestial: Security Service Management Architecture. http:// www.mcnc.org/ HTML/ITD/ ANR/ Celestial.html, March 1998. MCNC.

[14] D. Harkins and D. Carrel. RFC2409:The Internet Key Exchange (IKE), November 1998. http://www.ietf.org/rfc/rfc2409.txt.

[15] Abhijit N. Hayatnagarkar. On realizing traffic-driven security association establishment for ipsec. Master's thesis, Computer Science, North Carolina State University, may 1999. http://www.lib.ncsu.edu/etd/author/h.html.

[16] ICSA. Netlitmus: an anti-spoofing test tool, April 2000. http://www.icsa.net/html/communities/ddos/alliance/checker/index.shtml.

[17] F. Jou, F. Gong, C. Sargor, S. F. Wu, and R. Cleaveland. Architecture Design of a Scalable Intrusion Detection System for the Emerging Network Infrastructure. Technical Report E296, Adavnced Network Research, MCNC, April 1997.

[18] Y.F. Jou, F. Gong, S.F. Wu, H.Y. Chang, and et al. Design and implementation of a scalable intrusion detection system for the protection of network infrastructure. In *DARPA Information Survivability Conference and Exposition(DISCEX 2000)*. IEEE Computer Society press, January 2000.

[19] S. Kent and R. Atkinson. RFC 2401:Security Architecture for the Internet Protocol, November 1998. http://www.ietf.org/rfc/rfc2401.txt.

[20] S. Kent and R. Atkinson. RFC 2402:IP Authentication Header, November 1998. http://www.ietf.org/rfc/rfc2401.txt.

[21] S. Kent and R. Atkinson. RFC 2406:IP Encapsulating Security Payload (ESP), November 1998. http://www.ietf.org/rfc/rfc2401.txt.

[22] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. Internet Draft, IETF, draft-ietf-ipsec-arch-sec-04.txt, March 1998. Network Working Group.

[23] Catherine Meadows. A formal framework and evaluation method for network denial of service. In *Proceedings of the 1999 IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, June 1999. http://chacs.nrl.navy.mil/publications/CHACS/CRYPTOindex.html.

[24] Catherine Meadows. Open issues in formal methods for cryptographic protocol analysis. In *DARPA Information Survivability Conference and Exposition(DISCEX 2000)*. IEEE Computer Society Press, January 2000. http://chacs.nrl.navy.mil/publications/CHACS/CRYPTOindex.html.

[25] Jonathan Millen. A resource allocation model for denial of service. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, pages 137–147. IEEE Computer Society Press, 1992. http://www.csl.sri.com/ millen/index.html.

[26] RAVINDRA NARAYAN. Socket api extensions to extract packet header information list (phil). Master's thesis, Computer Science, North Carolina State University, may 1999. http://www.lib.ncsu.edu/etd/author/n.html.

[27] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson. Practical network support for ip traceback. Technical report uw-cse-2000-02-01, Department of Computer Science and Engineering, University of Washington, February 2000. http://www.cs.washington.edu/homes/savage/traceback.html.

[28] Dawn Song and Adrian Perrig. Advanced and authenticated marking schemems for ip traceback. Technical Report UCB/CSD-00-1107, Computer Science Department, University of California, Berkeley, June 2000. http://paris.cs.berkeley.edu/ dawnsong/research.html.

[29] Ion Stoica and Hui Zhang. Providing guaranteed services without per flow management. In *Proc. of the 1999 ACM SIGCOMM*, pages 81–94, October 1999.

[30] Robert Stone. Centertrack: An ip overlay network for tracking dos floods, October 1999. http://www.nanog.org/mtg-9910/robert.html.

[31] Van C. Van. A Defense Against Address Spoofing Using Active Networks. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 1997. http://www.sds.lcs.mit.edu/publications/van97.html.

[32] B. Vetter, F. Wang, and S.F. Wu. An Experimental Study of Insider Attacks for the OSPF Routing Protocol. In *IEEE International Conference on Network Protocols (ICNP)*, pages 293–300, October 1997.

[33] S.F. Wu, H.Y Chang, Y.F. Jou, and et al. Jinao: Design and implementation of a scalable intrusion detection system for the ospf routing protocol. *to appear in the Journal of Computer Networks and ISDN Systems*, 2000.

[34] Che-Fn. Yu and V. Gligor. A specification and verification method for preventing denial of service. *IEEE Transactions on Software Engineering*, 16(6):581–592, June 1990.

[35] J. Zao. Policy-Based Dynamic Security Management. http:// www.darpa.mil/ito/ Summaries97/F318_0.html, March 1998. BBN.