# Adaptive Use of Network-Centric Mechanisms in Cyber-Defense*

Michael Atighetchi, Partha Pal, Franklin Webber, Christopher Jones
matighet@bbn.com , ppal@bbn.com , fwebber@bbn.com, ccjones@bbn.com
BBN Technologies LLC

## Abstract

*Attacks against distributed systems frequently start at the network layer by gathering network related information (such as open TCP ports) and continue on by exhausting resources, or abusing protocols. Defending against network-based attacks is a major focus area in the APOD (Application That Participate in Their Own Defense) project, which set out to develop technologies that increase an application's resilience against cyber attacks. This paper gives an overview of APOD's current set of network-level defenses. Specific network-based defense mechanisms are described first, followed by a discussion on how to use them in local defensive behavior. Defense strategies, which specify coordinated defensive behavior across a distributed system, are discussed next, followed by results from initial experimental evaluation.*

## 1 Introduction

Defense enabling is an approach to make critical applications more survivable. In this approach, a defense strategy and supporting defense mechanisms are integrated with the application based on its survivability requirements. The concept of defense enabling as well as the process and toolkit supporting it were first developed in the DARPA APOD (Applications that Participate in their Own Defense) project [25, 35].

Several applications of varying complexity and survivability requirements were defense-enabled throughout the course of the project. Some network capabilities such as monitoring and manipulation of network traffic were required in most of these cases. Over time, parts of the custom network strategies were factored out of the application's integrated defense and packaged into stand-alone components within the QuO middleware [29, 26] framework.

This paper discusses network-based defenses and is organized as follows. Section 2 describes general motivations behind using network-based mechanisms in defense. Section 3 describes a set of network-based mechanisms used in various defense-enabled applications. Section 4 details how these mechanisms can be utilized to support defensive behavior which is local in scope. Local defenses can be combined and extended to form defense strategies, as explained in Section 5. Our experience in experimentally evaluating the utility of network-based defenses is described in Section 6. Section 7 concludes the paper.

*This work is funded by DARPA under contract number F30602-99-C-0188. The APOD toolkit is available as open source software from http://apod.bbn.com/release/latest.

## 2 Rationale for Network-Based Defense

It is generally accepted that attacks on computer systems will continue to happen, and some will be successful, because a truly secured infrastructure is impractical and costly, if not impossible. New attacks are constantly being devised and incorrect configurations and other flaws continue to exist in real systems. Therefore, it is crucial to detect the effects of successful attacks early on and take countermeasures to limit or repair the damage. Since the network is often the starting point for attacks, network-based defense is essential for *intervening early* in the attack while the attacker is still trying to gain a toe hold or performing reconnaissance. Defense mechanisms introduced in the network force attackers to work around them, which prolongs the application's useful life and increases attackers' visibility. Interrupting an attacker during his attempts to gain access to various parts of the system can potentially limit the parts he will be able to take over.

Continued operation of many applications depends in part on the availability of network resources, such as connectivity and bandwidth. Consequently, a major category of attacks is concerned with consuming or corrupting these resources. As an example, an attacker might flood network links with artificial load so that application packets cannot reach their destination in time. This results in unexpectedly long response times and other undesirable effects such as timeouts or exceptions. Various measures can be introduced to defend against network-resource depletion attacks. As part of its defense, the application may set up bandwidth reservations to get guaranteed throughput to essential ser-

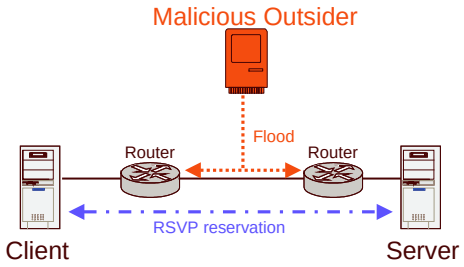


**Figure 1. Reservations limit effects of outsider floods**

vices. Reservations can proactively be put in place, or can be attempted on demand as a reaction to some observed delay or timeout.

Another common type of resource depletion is targeted at TCP stack resources. The attacker might launch a flood of connection requests to a service port and hold on to the connections as long as possible. Each connection acquires memory and CPU resources, and legitimate connection requests are denied once these resources are over-utilized. As part of its defense, the application may bound the number of parallel connections, raise alarms, and reject new connection requests from hosts which try to establish a large number of connections in a short amount of time.

*Direct manipulation of network elements*, such as routers and boundary controllers, can provide flexible and effective defenses against some denial of service attacks. Network devices are typically shared across multiple different applications; this is where defense enabling evolves from defending a single critical application to defending the aggregate system. Any defense implemented on the aggregate network (i.e., routers and boundary controllers) must take into account the effect it imposes on the whole system, consequently requiring more coordination. Examples include rate limiting and filtering at boundary controllers and the use of system wide bandwidth reservations to protect critical traffic.

Finally, an application's defense may simply involve better usage of existing security services, requiring *control of network related activities at the hosts*. For instance, defending an application may require that all communication between two key components must go over encrypted IPsec channels (as opposed to traditional IP). The change from IP to IPsec could be statically done at startup, or dynamically in response to some event, allowing one to specify a tradeoff between performance and security by only enabling costly encryption when the application is under attack.

## 3 Network-Centric Defense Mechanisms

This chapter describes the set of underlying network-based mechanisms used in APOD. A description of other capabilities, including filesystem, host, and replication management, can be found elsewhere [17, 34]. Many network-based mechanisms outlined in this section are open source tools, all of them have been used in defense enabling via the APOD toolkit.

- **Intrusion Detection** - Snort [28] is a lightweight signature-based intrusion detection system which tries to identify network attacks by matching traffic to a frequently updated set of attack patterns. Snort is capable of detecting portscans, well-known buffer overflow attacks, and trojan horses. Snort is used in many APOD applications as a network sensor.

- **Firewalls** - Netfilter [19] (also called Iptables) implements a stateful firewall on top of the Linux operating system. Using Iptables one can block certain traffic, redirect outgoing traffic to a tunnel, change IP source or target addresses via network address translation, and measure throughput via IP accounting. Iptables is typically used in APOD as a distributed firewall to block suspected traffic.

- **TCP Stack Probes** - Netstat [30] reports information about the internal state of the TCP stack. The number of parallel connections to a specific TCP port and the state of the connections are of particular interest to the defense, since many denial of service attacks work by exhausting TCP stack resources. In addition, the defense can obtain information about the MAC to IP address mapping by inspecting local ARP caches via the Arp command [8]. This command also provides the capability of statically specifying mappings to prevent ARP cache poisoning attacks [31].

- **Virtual Private Networks** - VPNs allow end-systems to establish a trusted communication channel over an untrusted network, e.g. the Internet, by employing encryption and authentication mechanisms. The APOD toolkit has interfaces to FreeS/WAN [6], a Linux implementation of IPsec [9], which allows applications to dynamically establish tunnels. In addition, lightweight user-space tunneling tools were also integrated, including openssh [23] and Zebedee [39].

- **Bandwidth Reservation Schemes** - Multiple applications often share the same network and thereby contend for available bandwidth. Intserv [36] and Diffserv [1] evolved as two alternative means to specify how bandwidth is assigned to traffic. Intserv, as represented by RSVP [2], employs a signaling protocol

to ensure end-to-end reservation attributes, requiring routers along the path to keep internal reservation state. Diffserv, in contrast, pushes the complexity to the network edges: traffic marking and conditioning takes place in border routers, allowing the core routers to treat packets based on their markings and eliminates the need to keep reservation state.

APOD experimented with two different implementations of RSVP to defend the system against outsider floods (see Figure 1) : A CORBA-wrapped version of the ISI implementation named quoRsvporb [27] and a security enhanced version of the University of Darmstadt implementation called SERSVP [38]. In addition, APOD prototyped a Bandwidth Broker [21], which provides a unified interface to the application for specifying its bandwidth requirements.

- **Traffic Shaping** - The Iproute2 [8] package enables Linux hosts to function as replacements for hardware routers. Intserv and Diffserv on Linux routers are made possible by full-fledged queue management support. Traffic shaping is implemented through a token bucket filter queue. The defense can specify a maximum forwarding rate and further refine the policy to allow short-term bursts.

# 4 Localized Defensive Behavior

Simple use of network-centric defense mechanisms involves reactive responses with local scope, utilizing capabilities of a small number of mechanisms. Such tactics tend to employ one mechanism for its sensor capability and tie it to a second mechanism for reaction. Figure 2 summarizes a sampling [1] of the tactics we have investigated in APOD.

Local tactics are highly reusable and self-contained. However, an individual localized response based on incomplete (i.e., local) knowledge may not by itself prolong the useful life of a distributed application. Nontheless, they are necessary components of an overarching defense strategy, and, as shown in Section 5, multiple local tactics can be combined in a coordinated way to mount an effective defense.

## 4.1 Reactive Tactics

The mechanisms described in Section 3 perform specific security related tasks. For example, Snort provides attack alerts, Iptables blocks traffic, and Iproute2 rate limits traffic. A defensive tactic attempts to integrate these services to mount local defensive responses.

[1] This is a non-exhaustive snapshot of our defense enabling work; more mechanisms will be added as we expand our technology base.

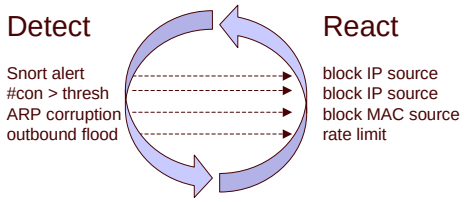


**Figure 2. Reactive tactics based on local knowledge**

The tactic against TCP connection floods, for instance, uses Netstat in combination with a threshold counter to sense TCP connection floods. It mounts a response by blocking traffic from the suspected source of the flood. This defense tactic is intended to react quickly, however, its actions may have a relatively high false-positive rate in the presence of spoofing.

In addition to the defense outlined above called *Choking TCP Connection Floods*, we have developed three more network-based reactive tactics named *Blocking Suspicious Traffic*, *Containing ARP Cache Poisoning*, and *Squelching Insider Floods*, which are described in more detail below.

Continuing with this idea in the ITUA project [4], we have encapsulated such reactive tactics in the form of *ITUA Rapid Reaction Loops*. Loops provide rapid reaction based on local knowledge to disrupt the attacker's flow early on in an attack. Changes caused by loops are limited in scope, often transient, and easy to reverse, which is especially important in the presence of false positives.

The remainder of this section describes some reactive tactics we experimented with.

### 4.1.1 Blocking Suspicious Traffic

**Attack Model**: Attackers often use portscans during reconnaissance to probe networks and determine what services are active on which hosts. As a next step, pre-scripted attacks against well-known vulnerabilities, such as buffer overflows, are frequently launched.

**Defense**: This tactic combines the Snort network intrusion detection system with the Iptables Linux firewall to block traffic to and from compromised machines. It contributes to defense in two ways. First, Snort is able to detect portscans an attacker might launch during reconnaissance. Disrupting these scans using Iptables to block traffic from the perceived source prevents the attacker from getting sensitive information. Second, Snort is capable of detecting many well-known signature-based attack patterns. Blocking such traffic prevents attacker hosts from taking part in future attacks.

**Limitations**: It is easy to spoof the source address of IP packets, especially if the attack does not rely on packets be-

ing send back to the source. Spoofing makes it possible for an attacker to trick the defense into blocking a wide range of essential servers. The fact that Snort's pattern matching rules are publicly accessible further exacerbates this issue by making it straightforward for an attacker to create a scenario in which different attacks seem to originate from multiple (critical) machines. Autonomic responses , such as blocking traffic, have little effect in this context and can even be harmful. A better strategy implies sending alerts to correlation intrusion detection systems, e.g., Emerald [20], which can then take actions based on a broader view or human input. A further limitation is given by the inherent race condition between ongoing attacks and the defense. If attacks are very fast, they might succeed before the defense tactic can detect them and reconfigure.

**Customization**: This defense tactic can either be deployed on end-systems or on LAN-wide boundary controllers, thereby scaling up the effects to a set of end-systems. In addition, one can specify a white-list of hosts that should never be blocked. The white-list is a simple means to prevent spoofing attacks from tricking the defense into blocking essential services.

**Alternative Approaches**: Shortly after this defense was developed under APOD, the hogwash project [7] started with the goal to use Snort as a traffic filter to throw out 95 percent of all scripted attacks. These development efforts were later folded back into Snort itself under the umbrella of "inline" Snort, which has recently become available. Our defense enabling technology can nicely accommodate such evolving mechanisms: shortcomings of an initial mechanism are usually compensated by middleware logic. As the underlying mechanism matures and starts to incorporate some of these (as shown by inline snort), the middleware implementation is refactored to take advantage of new capabilities in the mechanism.

#### 4.1.2 Choking TCP Connection Floods

**Attack Model**: Once the attacker has identified a service running on a dedicated TCP port, he might commence to establish many parallel connections to that port. Most servers create a socket for each connection and also spawn a thread to serve incoming requests, which leads to exhaustion of TCP and higher level protocol (such as RMI) stack resources in the presence of a large number of connections. This results in a denial-of-service situation, in which legitimate clients can no longer communicate with the service.

**Defense**: The defense tactic used to counter this attack continuously monitors the number of established TCP connections to a specific port via Netstat. When a preestablished maximum number of connections from a source is exceeded, it uses Iptables to reject further connections from the offending source for some period of time.

**Limitations**: An attacker can keep changing its source address and eventually achieve resource exhaustion since the connection threshold is source specific. Another attack approach may be comprised of spoofing the source of the attack so that the connection flood seems to originate from legitimate clients.

In the presence of spoofing, this tactic could lead to a self-inflicted denial of service. However, spoofing a 3-way TCP connection handshake is not as easy as spoofing single packet source addresses. Routers and VPNs, if configured properly, limit connection spoofing to IP addresses of the server's LAN.

**Customization**: Any existing Linux server application can be protected against TCP connection floods by simply deploying the APOD middleware component that implements this tactic. One needs to further specify what TCP port the service is listening on, how frequently the connections status is updated, and the maximum number of parallel connections. Running the defense component on client hosts instead of server hosts shifts the defense's focus from protecting servers to preventing clients from being used as launching pads for distributed denial of service attacks.

**Alternative Approaches**: Some mechanisms already provide means for dealing with connection floods. Iptables features transparent rate limiting of incoming TCP connections in terms of connections per second. The Zebedee tunneling mechanism employs the notion of connection timeouts; connections are closed after a certain time of inactivity to prevent connection flooding. In addition, systems designed with survivability in mind sometimes have similar strategies built into higher-level protocols. The Cougaar agent middleware [3] features built-in connection counting in its adaptive, QuO-managed transport layer [41].

#### 4.1.3 Containing ARP Cache Poisoning

**Attack Model**: In an ARP spoofing [31] attack, the ARP cache of a host in the same collision domain as the executing host is corrupted, which allows attackers to intercept traffic by changing ARP entries. ARP spoofing is a convenient method for sniffing through a switch (also known as active sniffing) [31]. The attack host can either drop packets to deny traffic, or forward traffic to the actual destination while doing stealth monitoring. A tool commonly used in such attacks is arpspoof [31].

**Defense**: The ARP spoof defense continuously monitors the mapping of MAC to IP addresses on a system using the arp command. Changes can be either due to legitimate reconfigurations (replaced NICs) or cache poisoning attacks. This tactic assumes that the defense would be reinitialized after valid reconfiguration, and therefore takes immediate action upon detecting cache changes by reseting it with the correct values. Furthermore, traffic to and from offending

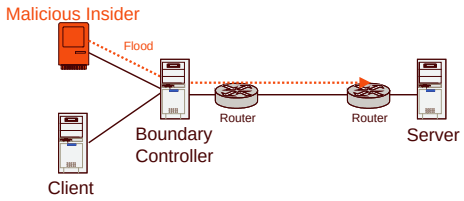


**Figure 3. Boundary controllers contain insider floods**

MAC addresses is blocked. Finally, an alert is passed to higher level decision making components.

**Limitations**: Since ARP cache poisoning cannot be executed across well-configured router boundaries, it is indicative of an attacker with direct access to a network segment. Because attackers have a large arsenal of attack tools which rely on direct network access, specific reactions to counter ARP cache poisoning might be of limited use and a higher level policy should consider to take more drastic actions, such as containing the whole LAN.

**Customization**: The tactic can be initialized with a correct mapping to start with or take a snapshot at startup time. Similar to previously described defenses, MAC addresses can be placed on a white-list to prevent them from being blocked and the monitoring rate can be adjusted. The defense component can run on end-systems, but the preferred insertion point is on boundary controllers.

**Alternative Approaches**: High quality switches often have a feature called MAC binding which prevents MAC addresses associated with ports from changing once they are initially set. Legitimate MAC changes can be performed by the network administrator on a per-case basis. On the downside, MAC binding greatly increases the overhead of the network operations team and is therefore highly frowned upon. Along the same lines, the use of static ARP entries is considered impractical in most cases for scalability reasons. Arpwatch [22] is a program with monitoring functionality similar to the defense outlined here. It "listens" for ARP replies on a network and stores a table of IP/MAC associations in a file. The tool notifies network administrators via email when changes occur.

#### 4.1.4 Squelching Insider Floods

**Attack Model**: If an attacker manages to take over a central application server, he might use network load generators to exhaust all available bandwidth between application components. A similar category of attacks involves propagation of malicious code (e.g. through trojan horses) to a large set of clients, using them as launching pads for distributed denial-of-service floods.

Figure 3 depicts a related case in which a malicious insider uses a non-application machine on the client network as a load generator. RSVP and Diffserv are only of limited value against such insider flooding attacks. Even if all communication between Client and Server is VPN protected, authentication is most often based on IP-identity, and therefore cannot distinguish between legitimate traffic and malicious floods from the same machine. In Figure 3, the virtual private network configured to create a tunnel between the two routers only protect against spoofing of addresses between the two routers, making it easy for an malicious insider to create floods which seem to originate from the client host.

**Defense**: The defense uses the notion of boundary controllers to contain floods. These IP-level firewalls guard each LAN and are injected into the communication path to the outside world. The flood containment tactic (running on a boundary controller) continuously monitors outgoing traffic via Iptables and calculates throughput metrics including packets/second and bits/second. During a calibration phase, expected mean values are saved internally and used as a baseline during normal operation. Comparisons of means between observed and expected parameters are executed in regular intervals to determine whether the observed outgoing traffic is significantly higher than expected. If so, routing configurations at the boundary controllers are changed to rate limit outgoing traffic via a token bucket filter. This special queuing filter creates a virtual bucket based on a description of maximum allowable throughput in bits per second. The Linux routing code places incoming packets in the bucket, which "leaks" forwarded packets at the specified rate. Once the bucket is full, it starts dropping packets. Rate limiting is kept in place for a certain period of time, after which the defense reverts back to the startup configuration.

**Limitations**: The current implementation relies on comparison of means to detect floods. Alterations in the traffic patterns which do not change the mean therefore go undetected. An attacker might impose a new traffic pattern where outgoing traffic cycles through phases of maximum and zero output, not changing the mean. A time synchronized combination of multiple such cycles in different LANs could easily exhaust central network resources. Along similar lines, it might not be possible to gather meaningful calibrated mean values due to the highly dynamic nature of some applications. Both false positives and undetected floods could result from biased calibration. Finally, this defense strategy requires a boundary controller to be added to a LAN in order to contain floods, which increases both run-time and physical overhead.

**Customization**: Boundary controllers are placed as guards at the edges of a security domain. If a host has multiple security domains, for instance by virtue of hosting multiple virtual machines [32] on a security enhanced operat-

ing system [13, 14], the defense component can be installed on the base host to contain floods originating from the virtual machines. LANs form network security domains (also called collision domains) by virtue of sharing resources within the domain. A separate LAN-based boundary controller is used as a guard in this case. For detecting floods, parameters like measurement frequency, calibration window size, observation window size, data points to drop upon startup, and significance interval for hypothesis tests can be adjusted to suit application specific needs. Customizable actuator parameters include maximum throughput, maximum short term burst throughput, and rate limiting period.

**Alternative Approaches**: Best practices in network design [5, 11] offer patterns to contain floods. Clear separation of collision domains and bandwidth over-provisioning are some examples of effective network design. Given a certain number of client server communication paths, the impact of remote floods on the server is correlated to the size of the pipe between the flooding client and the server: A single client connected to a server over the Internet is likely to only deny access to legitimate clients in its topology neighborhood. However, distributed denial of service attacks are often comprised of many small coordinated floods, effectively bringing down large parts of server access networks. D-WARD [16] proposes a mechanism in which source routers detect and rate limit suspicious flows. D-WARD features fine-grained protocol-specific statistics and determines the rate limit by exponential correlating to attack flow rates. Edge routers in Diffserv [21] have capabilities comparable to APOD's boundary controllers, but are often statically configured via pre-negotiated service level agreements. VPNshield [33] provides an integrated solution to contain outsider floods, whereas the APOD defense tactic discussed here deals with insider floods.

#### 4.1.5 Evading Port Attacks through Port and Address Hopping

**Attack Model**: Traditional security mechanisms try to thwart attackers by encrypting and encapsulating data packets. The encrypted data is then sent between two dedicated fixed endpoints. This leaves outside attackers with the possibility of detecting endpoints via traffic analysis. For inside attackers, who can usually obtain port information via sniffing the communication after it has been decrypted, getting IP information is even easier. The next step in many attacks is to run well-known pre-existing exploits.

**Defense**: Port/Address hopping is a dynamic tactic that constantly changes a service's TCP identity, i.e., its IP address and TCP port. The intention is to both hide the service's real identity and confuse the attacker during reconnaissance.

In TCP communications, all messages exchanged between two parties contain a source and destination (address:port) pair. In addition, higher-level DOC protocols, such as IIOP, embed TCP information in object references. IP Port hopping continuously replaces the source and destination port with randomly picked numbers and redirects traffic accordingly. Address hopping goes one step further by randomly changing the address part. Packets intercepted by attackers will reveal random addresses, which are valid only for a small period of time, e.g., 1 minute. For a port attack to be successful, the attacker must discover the current ports and execute the attack all within one refresh cycle.

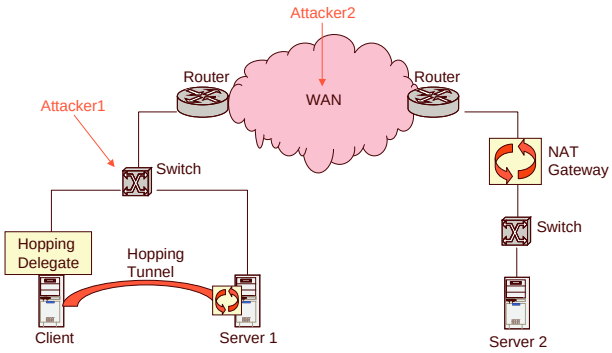


**Figure 4. Two designs for implementing port and address hopping**

As additional benefit, this tactic increases the likelihood of an attacker to be detected. An attacker who is not aware of the dynamic changes might run attacks against "stale" addresses and ports, raising alerts. A more sophisticated attacker might try to predict future selections based on addresses and ports observed during reconnaissance. Since the defense chooses addresses and ports randomly within a certain range, the attacker's predication is likely to be wrong, raising further alerts.

Figure 4 displays the major components and their usage. The hopping mechanism is implemented by a client component called hopping delegate and network address translation (short NAT) gateway. The hopping delegate is directly located on the client machine (or even in the client's process in some cases). It intercepts IIOP RPC calls to the real server, and replaces all (realaddress:realport) header information with (fakeaddress:fakeport).

The NAT gateway is located either on the server's LAN or directly on the server host. It does the reverse mapping from (fakeaddress:fakeport) to (realaddress:realport). The (fakeaddress:fakeport) pair is picked randomly from a range of IP addresses and ports. It is used for a specific cycle time, after which a new pair is generated and used. Information about the previous pair is saved to identify suspicious traffic using stale pairs.

Note that this mechanism relies on synchronization of

random number generators between the two components which can be achieved by seeding both generators with the same initial value. In addition, time synchronization is required to coordinate the switchover using a newly generated (fakeaddress:fakeport) pair. We identified the following two use cases, which lead to two different designs and implementations:

- **Tunnels** - Client and server have addresses within the same IP network. The communication between Client and Server1 in Figure 4 does not involve any routers, since both machines are on the same IP network and connected via a switch. In order to prevent Attacker1 from detecting Server1's port, the client's delegate redirects traffic to go over a tunnel to Server1, changing server side ports (and also possibly client side ports) of the tunnel randomly over time. In this scheme, APOD's adaptive defense is limited to port hopping.

- **NAT Gateway** - Client and server are located on different IP networks. For communication between Client and Server2, the client component changes the target address of packets destined for Server2 to a random IP address within the same IP network as Server2. In addition, a random port is selected. This packet is routed to the NAT gateway, which forwards the packet to Server2. Replies from Server2 to Client are again forwarded through the NAT gateway to have their source IP address and port adjusted to the random selection. Both Attacker1 and Attacker2 see only packets between Client and random IP addresses and random ports. If required, an equivalent NAT gateway on the client's LAN can obfuscate Client's IP address and port, so that Attacker2 is only able to see traffic between random addresses and ports of hosts in the two LANs.

**Limitations**: For each system, the choice of random ports is limited by the number of legitimate TCP ports minus the ports already used up by other services. Legitimate TCP ports are typically in the range between 1024 and 65535 on both client and server systems, which leaves a range of roughly 64000 numbers for hopping purposes, assuming that only a small number of ports are used up by other TCP connections. Furthermore, address hopping is limited by the number of routable host addresses within the client's and server's IP address space, since packets with fake addresses still have to reach the NAT gateway. Dynamically changing ports and addresses might be impractical for servers protected by COTS hardware firewalls, since it requires the firewall to sequentially open a wide range of ports. However, closer integration between the NAT gateway and the firewall (e.g., via SSH controls) might mitigate this problem: The gateway could send authenticated signals to the firewall indicating which dedicated port to open at any given point in time.

**Customization**: In addition to the two different deployment modes described above, one can customize various parameters, including:

- the range of port numbers (and certain ports in that range can be excluded),

- how long a selection of IP identities should be used until the next one is picked randomly, and

- how long an old selection of IP identities should stay active to smooth transition behavior.

Various tunneling mechanisms, including SSH, Netcat, and Zebedee, have been integrated and can be selected upon startup time.

**Alternative Approaches**: The DYNAT project [10] has implemented and validated an approach similar to the NAT gateway. Compared to DYNAT, the APOD tactic provides hopping functionality on protocol layers above TCP, such as distributed CORBA calls, which requires additional modification of TCP/IP data in the IIOP protocol. Furthermore, the APOD solution relies on standard COTS utilities such as Linux Iptables and Zebedee tunnels to implement the desired functionality, whereas DYNAT is a more hardware integrated, specialized solution.

CONTRA [12] implements IP address dispersion by adding CONTRA headers containing the real destination to packets. The actual packet addresses are then transformed and forwarded over a set of relay hosts to the final target. The relay operation includes decryption of the CONTRA header, extraction of the real destination, changes to the padding, and re-encryption with the key of the next hop. Compared to the DYNAT or APOD, CONTRA requires changes to the routing infrastructure (i.e. relay hosts) to support the CONTRA protocol.

## 5 Defense Strategies

The individual adaptive responses described in Section 4 are not enough for effectively defending critical applications. Multiple tactics are required to address an application's survivability requirements. Figure 5 shows how APOD allows one to combine individual mechanisms and tactics into higher-level defense strategies.

The local defense tactics described in Section 4.1 are built on top of mechanisms and rely on local knowledge. Individual tactics are therefore vulnerable to certain attacks (e.g. spoofing) that try to trick the defense into inducing self-inflicted denial of service. Distributing such isolated weak responses might further force this issue. In addition,

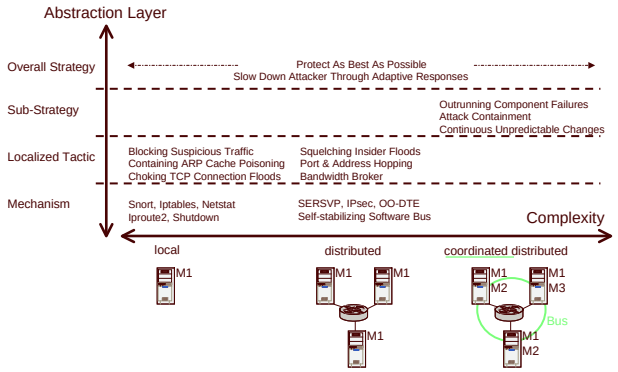


**Figure 5. Integration of mechanisms and local tactics in an overall defense**

tactics focusing on different aspects of the system need to coordinate to successfully work together. For example, port hopping frequently changes the destination ports of packets, which could be interpreted by Snort as a port scan in absence of coordination between the two mechanisms. Therefore, more sophisticated defense strategies involve coordination among constituent (sub-)strategies to build an overall defense. We describe an example of such an overarching strategy in this section.

The basic objective of the top-level strategy is to increase an application's survivability through more coordinated defense behavior. It assumes that the existing infrastructure, including operating systems and networks, is *hardened* to a reasonable degree using COTS security tools. This means that application traffic is likely to be encrypted and authenticated via VPNs, prioritized using network reservations, and controlled via firewalls. In addition, best practices in network design are assumed, resulting in a structured network which is well provisioned to deal with expected traffic loads, has cleanly isolated collision domains, and may have a buffer zone to the outside world.

Although these approaches are rather binary and static in nature (either they are successful in providing security or they fail completely), they form the first line of defense. Defense enabling builds more dynamic responses on top of such a hardened foundation.

We have identified the following three defense substrategies:

- *Outrunning Component Failures*, which replicates key application components and intelligently places new replicas on suitable hosts upon noticing failures.
- *Attack Containment*, which isolates host intrusions and network based distributed denial of service attacks and stops their propagation.
- *Continuous Unpredictable Changes*, which tries to put strict time constraints on the usefulness of obtained attack information by constantly changing unpredictably.

The implementation of the sub-strategies is based on the QuO adaptive middleware framework [40], which provides architectural support and tools for encapsulating strategies in self-contained reusable components called qoskets [29].

The remainder of this section discusses each strategy and the corrresponding qoskets.

## 5.1 Outrunning Component Failures

The self-stabilizing software bus is a lightweight APOD mechanism for tolerating crash failures. The outrun strategy interfaces with the bus and, upon detecting a replica death, selects a new host for the replacement replica. The new host is picked from a list of possible candidates by searching for a host which is hard to infiltrate for the attacker based on the defense's current knowledge of the attack state. A host is preferred if it has not been infiltrated yet and is not expected to be shutdown soon (as an action of the containment strategy described below). In addition, the strategy gives higher preference to hosts located in a different network security domain, i.e., on a different IP subnet, relative to the observed fault. Further network centric information, for instance whether a network level intrusion has been observed on the candidate or its network, is also considered. This strategy is very agile with reaction times of a couple of seconds.

The reactive outrunning strategy was deployed and tested in a set of APOD Red Team experiments (which are further described in Section 6). A proactive mode was added after the experiments, in which the strategy keeps moving replicas from host to host, effectively keeping address and port information dynamic and hard to determine.

## 5.2 Attack Containment

The containment strategies described in this section are used to to prevent an attacker from being able to attack large parts of the system by taking over a single part of it. In the *Quarantine Hosts* strategy, individual machines are treated as security domains. This prevents an attacker who gains root privileges on one essential server from easily spreading the same attack to remaining servers. Containment qoskets run on all application hosts and coordinate their actions using the self-stabilizing software bus. The qoskets implement the *Block Suspicious Traffic* strategy combined with coordinated logic to consistently block attacks and prevent self-inflicted denial of service.

Upon detecting a port attack on host X, the qosket checks whether it is in a denial-of-service mode or experiencing

a common mode failure by checking how many hosts had been infiltrated by the same attack before. In the absence of denial of service attacks, the source of the attack packet is noted and communicated to all other qoskets. Upon receiving the attack notification, the qosket on host X tries to initiate an automatic *host shutdown* to prevent the infiltrated system from being used as a launch pad for further attacks. In addition, the qoskets on all other hosts change firewall rules to block traffic to and from X. To prevent self-inflicted denial of service, the strategy only contains up to a certain percentage of machines. After exceeding the limit, it only issues recommendations to the outrunning strategy which in turn avoids starting new replicas on infiltrated hosts.

To contain insider floods, the *Squelch Insider Floods* strategy is deployed on boundary controllers on each LAN. Since this defense only protects against floods originating from a LAN protected by the boundary controller, SERSVP is used in addition to defend against external floods of inter-LAN links. Bandwidth reservations are statically initiated upon startup. Finally, the boundary controllers are linked with host defenses to implement a *Trace Back Containment* policy. If a boundary controller gets notified that a host H within its domain has been marked suspicious, it will block traffic to and from H closest to H's source.

### 5.3 Continuous Unpredictable Changes

Sophisticated attackers often spend a long time trying to map the target network and gather as much information as possible about essential services. Under the assumption that determined attackers will get any information given enough time, keeping sensitive information about the system's state static does not make sense. The defense strategy described here tries to render information obtained by attackers harmless by changing it frequently. As an added bonus, the use of stale information raises alerts, which can serve as input to intrusion detection systems.

To maximize the defense, changes should be unpredictable for the attacker. Timeliness is a central aspect of this defense; on one hand, a high rate of change results in high overhead; on the other hand, infrequent change might allow an attacker to gather information and execute and attack within one refresh cycle. The strategy has to be flexible enough to allow one to describe this trade-off upon deployment. Examples of this strategy include port and address hopping (4.1.5), unpredictable server selection for client-server interactions, and unpredictable network route selection for connections.

Although this strategy is proactive, a reactive version may make sense if the defense can survive the attack at hand. In a replicated system, relocating replicas proactively might have unnecessary overhead compared to starting a new replica in a randomly selected clean domain only upon noticing a fault. Section 5.1 shows an example of reactive behavior which relies on properties of the underlying replication mechanism to tolerate replica faults.

## 6 Experimental Validation

The evaluation of network-centric defense took place in multiple stages throughout the project ranging from logical analysis to red team experimentation. Early in the project, a set of defense-enabled applications were subjected to internal red team testing by a developer outside of the APOD development team. This early experimentation clearly pointed out the need to address network-based attacks. New defense tactics including port hopping and blocking of suspicious traffic were developed, and strategies like containment and continuous reconfiguration were further refined.

Towards the end of the APOD project, the FTN/DC continuous experimentation program [15, 37] conducted two formal Red Team experiments to evaluate the maturity and applicability of APOD's dynamic defenses. Results of the experiments have been published elsewhere [18, 17, 24], including definitions of hypotheses, flags, rules of engagements, metrics, data analyses, and attacks.

Despite the fact that the Red Team was able to capture the denial flag most of the time, the experiments demonstrated that APOD improved survivability at an acceptable cost. The Red Team attempted to use the defense against itself to capture the flags. Attacks that were successful were multi-staged, composed of sub-attacks, and executed in a coordinated way, with each sub-attack aimed at achieving a partial goal towards capturing the flag.

## 7 Conclusion

Adaptive use of network-based capabilities is key to successful and effective defense. Defense enabling with network-based capabilities has demonstrated potential in forcing even highly skilled attackers to work hard to achieve their objective.

Network-based mechanisms and tactics are especially useful in enforcing containment and isolation at the network level. Containment attempts to keep the attacker from affecting wider parts of the system. Isolation attempts to separate useful traffic from suspected traffic with the hope that useful traffic can still be allowed. Network-centric responses can be extremely powerful and affect a large part of the system, therefore strategies using these must be carefully evaluated.

We have demonstrated that network-based responses can be utilized in dynamic defense, but further research is needed to develop new tactics, integrate new mechanisms, and evaluate the usefulness and cost-benefit tradeoffs before such defenses can be effectively deployed.

## References


[1] S. Blake et al. An architecture for differentiated services. Technical Report RFC 2475, Internet Engineering Task Force, http://ietf.org/rfc/, Dec. 1998.

[2] R. Braden et al. Resource reservation protocol (rsvp) version1 - functional specification. Technical Report RFC 2205, Internet Engineering Task Force, http://ietf.org/rfc/, Sept. 1997.

[3] Cougaar home page. http://www.cougaar.org.

[4] M. Cukier et al. Providing intrusion tolerance with itua, June 2002.

[5] P. Ferguson and D. Senie. Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing. Technical Report RFC 2827, Internet Engineering Task Force, http://ietf.org/rfc/, May 2000.

[6] FreeS/WAN. Free secure wide area network under linux 2.4. http://www.freeswan.org.

[7] Hoghwash. http://hoghwash.sourceforge.net.

[8] B. Hubert et al. Linux advanced routing and traffic control howto. http://ds9a.nl/2.4Networking/.

[9] Internet engineering task force (ietf) ip security charter. http://www.ietf.org/html.charters/ipsec-charter.html.

[10] D. Kewley, R. Fink, J. Lowry, and M. Dean. Dynamic approaches to thwart adversary intelligence gathering. In *DARPA Info. Survivability Conf. and Expo.*, May 2001.

[11] T. Killalea. Recommended internet service provider security services and procedures. Technical Report RFC 3013, Internet Engineering Task Force, http://ietf.org/rfc/, Nov. 2000.

[12] J. Lepanto and W. Weinstein. Contra - camouflage of network traffic to resist attack. In *DARPA OASIS PI Meeting*, 2002.

[13] P. Loscocco and S. Smalley. Integrating flexible support for security policies into the linux operating system. In *USENIX Annual Technical Conference*, 2001.

[14] P. Loscocco and S. Smalley. Meeting critical security objectives with security-enhanced linux. In *Ottawa Linux Symposium*, 2001.

[15] J. Lowry and K. Theriault. Experimentation in the ia program. In *DARPA Info. Survivability Conf. and Expo.*, May 2001.

[16] J. Mirkovic, G. Prier, and P. Reiher. Attacking ddos at the source. In *IEEE Int'l Conf. on Network Protocols*, 2002. to appear.

[17] B. Nelson, W. Farrell, M. Atighetchi, J. Clem, L. Sudin, M. Shepard, and K. Theriault. Apod experiment 2 - final report. Technical Report Technical Memorandum 1326, BBN Technologies LLC, Sept. 2002.

[18] B. Nelson, W. Farrell, M. Atighetchi, S. Kaufman, L. Sudin, M. Shepard, and K. Theriault. Apod experiment 1 - final report. Technical Report Technical Memorandum 1311, BBN Technologies LLC, May 2002.

[19] Netfilter. Firewalling, nat and packet mangling for linux 2.4. http://www.netfilter.org.

[20] P. G. Neumann and P. A. Porras. Experience with EMERALD to date. In *First USENIX Workshop on Intrusion Detection and Network Monitoring*, pages 73–80, Santa Clara, California, apr 1999.

[21] K. Nichols, V. Jacobson, and L. Zhang. A two-bit differentiated services architecture for the internet. Technical Report RFC 2638, Internet Engineering Task Force, http://ietf.org/rfc/, July 1999.

[22] N. R. G. of the Information and C. S. D. at Lawrence Berkeley National Laboratory. Arpwatch. http://www-nrg.ee.lbl.gov/.

[23] OpenSSH. open-source ssh. http://www.openssh.org.

[24] P. Pal, M. Atighetchi, F. Webber, R. Schantz, and C. Jones. Reflections on evaluating survivability: The apod experiments. In *IEEE Int'l Symp. on Network Computing and Applications (NCA-03)*, 2003. submitted.

[25] P. Pal, F. Webber, et al. Defense enabling using advanced middleware: An example. In *MILCOM*, Oct. 2001.

[26] P. Pal, F. Webber, J. Zinky, R. Shapiro, and J. Megquier. Using qdl to specify qos aware distributed (quo) application configuration. In *IEEE Int'l Symp. Object-Oriented Real-Time Distributed Comp.*, Mar. 2000.

[27] QuO. Quality objects open-source release. http://quo.bbn.com/release/latest.

[28] M. Roesch. Snort - lightweight intrusion detection for networks. In *USENIX LISA*, 1999.

[29] R. E. Schantz, J. P. Loyall, M. Atighetchi, and P. P. Pal. Packaging quality of service control behaviors for reuse. In *IEEE Int'l Symp. Object-Oriented Real-Time Distributed Comp.*, Apr. 2002.

[30] E. Siever et al. *Linux in a Nutshell*. O'Reilly, 2000.

[31] E. Skoudis. *Counter Hack*. Prentice Hall PTR, 2002.

[32] VMware. http://www.vmware.com.

[33] VPNshield. Technology for protecting vpns from denial-of-service (dos) attacks. http://www.atcorp.com/products, 2002.

[34] F. Webber, P. Pal, M. Atighetchi, and C. Jones. Apod final report. Technical Report Technical Memorandum, BBN Technologies LLC, Oct. 2002.

[35] F. Webber, P. Pal, et al. Defense-enabled applications. In *DARPA Info. Survivability Conf. and Expo.*, May 2001.

[36] P. White. RSVP and integrated services in the internet: A tutorial. *IEEE Communications Magazine*, pages 100–106, May 1997.

[37] B. Wood and R. Duggan. Red teaming of advanced information assurance concepts. In *DARPA Info. Survivability Conf. and Expo.*, Jan. 2000.

[38] T.-L. Wu et al. Securing QoS: Threats to RSVP messages and their countermeasures. In *Int'l Workshop on Quality of Service*, June 1999.

[39] The zebedee secure ip tunnel homepage. http://www.winton.org.uk/zebedee.

[40] J. Zinky, D. Bakken, and R. Schantz. Architectural support for quality of service for CORBA objects. *Theory and Practice of Object Systems*, 1(3):55–73, Apr. 1997.

[41] J. Zinky and R. Shapiro. Adding qos adaptation to component-based middleware. In *Int'l Conf. Distributed Comp. Syst.*, 2003. submitted.