



Adaptive Cyberdefense for Survival and Intrusion Tolerance

While providing some resistance against cyberattacks, current approaches to securing networked and distributed information systems are mainly concerned with static prevention measures. For example, signature-based systems can only detect known attacks and tend to provide brittle, all-or-nothing protection. New work in survivability and intrusion tolerance focuses on augmenting existing information systems with adaptive defenses. A middleware-based survivability toolkit lets applications use network- and host-based mechanisms in their own defense.

Much of 21st century life, including national defense and security, now depends on networked and distributed computing systems. An unfortunate side effect of this is that sensitive data and the critical processes consuming or producing such data are susceptible to attack.

The history of cybersecurity shows that flawless intrusion prevention and perfectly accurate intrusion detection are practically impossible. Approaches that attempt to prevent intrusions altogether result in expensive, noninteroperable systems. Even state-of-the-art intrusion-detection systems (IDSs) are ill-equipped to detect novel attacks and often give false positives and false negatives. Therefore, we must assume that some intrusions will succeed, at least partially, and that we won't detect some before their ill

effects manifest. Given this, survival seems a more practical and achievable goal of cyberdefense – that is, a defensive system must be able to operate through attacks, even if intruders have taken over major parts of the system.

Our work in survivability and intrusion tolerance centers around *defense enabling*.^{1,2} The DARPA Applications that Participate in their Own Defense (APOD) project^{3,4} has developed a middleware-based survivability toolkit that lets an application, in conjunction with its underlying infrastructure, respond to attacker actions based on a defense strategy determined by its survivability requirements. (See the “Related Work” sidebar for other research in this area.) Our approach for developing defense strategies involves combining a system's key capabilities with defensive tactics.

**Michael Atighetchi,
Partha Pal,
Franklin Webber,
Richard Schantz,
Christopher Jones,
and Joseph Loyall**
BBN Technologies

Related Work in Intrusion Tolerance

Since we first showed the feasibility of integrating defensive adaptation into an application in 1999, many researchers have explored various ways to use adaptive techniques in cyberdefense, and autoadaptive capabilities are now almost a standard feature in many intrusion-tolerant systems.

The Willow architecture¹ achieves intrusion tolerance using a combination of disabling vulnerable network elements when a threat is detected or predicted, replacing failed system elements, and reconfiguring the system if nonmaskable damage occurs. Willow uses control loops to perform proactive and reactive control actions that are similar to the adaptive responses in APOD. Willow has taken a more centralized position than APOD by having a dedicated analysis and diagnosis component and reconfiguration scheduler. APOD, in contrast, has a distributed management of adaptation and defense strategy.

Dependable Intrusion Tolerance (DIT)² comprises functionally redundant HTTP commercial-off-the-shelf servers running on diverse operating systems and platforms, hardened intrusion-tolerant proxies

that mediate client requests and verify the behavior of server and other proxies, and monitoring and alert-management components based on the Emerald³ IDS. The system adapts its configuration dynamically in response to intrusions and other faults. As with APOD, DIT has a range of adaptive responses to isolate compromised parts in a system.

Malicious and Accidental Fault Tolerance for Internet Applications (MAFTIA)⁴ is a European project developing an open architecture for transactional operations on the Internet. MAFTIA models a successful attack on a security domain, leading to corruption of processes in that domain, as a fault; the architecture then exploits approaches to fault tolerance that apply regardless of whether the faults are due to accidents or malicious acts. MAFTIA is explicitly middleware-based and provides both protection from and tolerance of intrusions. Unlike APOD, MAFTIA allows for Byzantine behavior in the system.

The Saber⁵ system, like APOD, combines several mechanisms into a coherent defense. The Saber defense mechanisms include intrusion detection, automatic code patching, process migration, and filtering of

distributed denial-of-service floods. Saber concentrates on defending the server side of client-server applications on the open Internet, whereas APOD aims to defend applications end to end, including clients.

References

1. J. Knight et al., "The Willow Architecture: Comprehensive Survivability for Large-Scale Distributed Applications," *Proc. Int'l Conf. Dependable Systems and Networks (DSN 02)*, supplemental vol., IEEE Press, 2002, pp. C.7.1–C.7.8.
2. A. Valdes et al., "An Architecture for an Adaptive Intrusion Tolerant Server," *Proc. Security Protocols Workshop*, Springer-Verlag, 2002, www.sdl.sri.com/papers/pl/r/protocols_SRI_02/protocols_SRI_02.pdf.
3. P.G. Neumann and P.A. Porras, "Experience with EMERALD to Date," *Proc. 1st Usenix Workshop on Intrusion Detection and Network Monitoring*, Usenix Assoc., 1999, pp. 73–80, www.csl.sri.com/users/neumann/det99.html.
4. P. Verissimo, N.F. Neves, and M. Correia, "The Middleware Architecture of MAFTIA: A Blueprint," *Proc. 3rd IEEE Information Survivability Workshop*, IEEE CS Press, 2000, www.navigators.di.fc.ul.pt/docs/abstracts/isw00.html.
5. A. Keromytis et al., "A Holistic Approach to Service Survivability," *Proc. ACM Workshop on Survivable and Self-Regenerative Systems*, ACM Press, 2003, pp. 11–20.

Adaptive Cyberdefense

Distributed and networked systems are susceptible to attacks ranging from those mounted by "script kiddies" to sophisticated cyberterrorists and nation states. A significant number of such attacks involve some kind of intrusion from the outside that gives attackers a toehold in the system from which they can either achieve their objectives directly or expand their privileges to a level that will let them do so.

We use the term *intrusion* as researchers use it regarding IDSs, implying that attacks that start with a high level of privilege that doesn't need to be expanded — for example, attacks by physical access or social engineering — are out of bounds.

Defense enabling's goal is to defend sensitive distributed and networked applications from outside cyberattackers who are intent on disrupting the application's operation. Often, such applica-

tions are in private domains, offering no publicly available services to the open Internet community. They often run inside restricted perimeters (such as various classes of military networks) and in controlled environments with well-known sets of other applications. These critical applications form high-value targets, with potentially devastating effects upon failure.

Adaptation plays a key role in surviving intrusion attempts to disrupt an application by consuming or corrupting the resources it needs to continue. If the system doesn't adapt to malicious changes in the environment caused by intruders, it becomes an easier stationary target for attack. Such adaptation requires strategic integration and synergistic interoperation of various mechanisms such as IDSs, host and network probes, firewalls, virtual private networks (VPNs), process and data replicators, host controllers, and other resource managers.

Organizing Adaptive Cyberdefense in Middleware

Our goal is to harden a distributed application, making it more agile and resistant to a set of malicious attacks. We must assume that attackers will be able to exploit some security flaws in the systems and networks on which the application is built, thus gaining privileges on some network nodes. On these nodes, they can consume resources and potentially modify the local behavior of the application's components. If the application runs on Linux operating system nodes, for example, attackers could exploit a Linux flaw that allows "root" privilege and then kill all application processes on those nodes and use them to flood other nodes with network traffic. A defense-enabled application will adapt so that it can continue working despite such attacks.

Because an attack can cause application components to fail, defense enabling depends in part on fault-tolerance techniques.⁵ The application should be distributed redundantly over several heterogeneous nodes, under the assumption that the attacker is unlikely to be able to exploit security flaws in all of them simultaneously, so some application components will continue to work correctly on at least some nodes.

The first step in defense enabling is to identify which potential attacks are most likely for the application in its domain. For example, the APOD toolkit assumes that the attacker will be able to kill processes but will be unlikely to cause arbitrarily corrupt behavior. (Later work on defense enabling has let us remove this assumption when necessary.⁶)

The next step is to develop a defense strategy, focusing on the set of likely attacks. We can decompose defense strategies into substrategies and local tactics. Every strategy will depend on a set of defense mechanisms chosen to thwart or counter a particular kind of attacker behavior and will coordinate the mechanisms into a coherent defense.

The final step is to integrate the defense mechanisms so that they coordinate as directed by the defense strategy. We can integrate the mechanisms with code in the application, but in most cases, the coordination code is better placed in a middleware layer. Implementing the defense strategy in middleware gives a clean separation between the application's functionality and the additional, quality-of-service (QoS) properties added by the defense. This separation allows engineers who aren't the application's developers to build the defense, and it facilitates reusing similar strategies

in different applications.

The APOD toolkit is built using the Quality Objects (QuO)⁷ middleware-based capability for managing dynamic QoS-oriented behaviors. (Open-source software is available from <http://quo.bbn.com> and <http://apod.bbn.com>.) QuO makes an application resource-aware and able to adapt to changes in the environment – precisely the features needed in a defense strategy. QuO applications can specify QoS requirements (such as performance and bandwidth), measure the level of QoS that is being provided, and adapt by invoking resource managers that affect the QoS level. QuO provides a set of high-level quality-description languages (QDLs) to specify QoS and describe adaptation algorithms. QDLs are at the same level of abstraction as interface description languages in the distributed-object computing paradigm.

We have used QuO to provide adaptation to meet real-time constraints and control bandwidth allocation.⁸ We have also formalized several patterns of adaptive middleware use⁹ and used aspect-oriented techniques to insert adaptive behavior into existing applications.¹⁰

Defenses and Their Hierarchical Decomposition

The basic objective of a defense strategy is to increase an application's survivability through automated coordinated defensive behavior. We assume the strategy will be used in conjunction with techniques that harden the existing infrastructure, including operating systems and networks, using state-of-the-art protection technologies. This hardening (or protection) forms the first line of defense. Defense enabling builds dynamic responses on top of a hardened foundation, assuming the availability of many key network technologies, including the IP Security Protocol (IPsec), VPNs, Secure Shell (SSH) tunnels, firewalls, and Resource-Reservation Protocol (RSVP) reservations.

High-Level Strategies

The overall strategy is to significantly improve the first line of defense by managing the shortcomings and failures of the protection it provides. Our current approach comprises three substrategies.

First, we can *outrun component failures* by replicating key application components and intelligently placing new replicas on suitable hosts when components fail. Replicating components lets the system continue operating even if some of the replicas fail. Consumers of data from the repli-

cated components are unaffected by a replica's loss: the defense system masks component failures by restarting new replicas to replace them. The outrun substrategy contains an algorithm for determining the next-best host to restart replicas on. This selection algorithm gives a higher rank to hosts that have a different operating system than the last infiltrated host or are located on a different LAN. The algorithm also considers additional network-centric information, such as whether network-level intrusions have been observed previously by sensors on the candidate or its network. This substrategy depends on the premise that at least some components can be replaced at a faster rate than an attacker can remove them.

Second, we attempt to achieve *attack containment* by isolating host intrusions and network-based distributed denial-of-service attacks to stop their propagation. Attackers tend to favor attacks that let them infiltrate large parts of a system by taking over only a small part of it. To counter such threats, this strategy lets us specify multiple fine-grained security domains that split the system into independent privilege domains. The adaptive defenses then enforce each domain's policy and, upon detecting policy violations, might decide to isolate the offending domain in an effort to cut off the attacker.

Lastly, we place strict time constraints on information's usefulness by performing *continuous unpredictable changes* to configuration parameters. Sophisticated attackers often spend significant effort gathering information about essential services. Assuming that determined attackers will get any information given enough time, it pays to render disclosed information useless by changing it frequently. An added benefit is that attackers end up alerting IDSs by using stale information.

Timeliness is a central aspect of this defense. On one hand, a high rate of change increases overhead, but infrequent change could let an attacker gather information and execute an attack within one refresh cycle. The APOD toolkit is flexible enough to express this trade-off at deployment time.

Combining this defense with the reactive outrunning-component-failures defense yields a proactive variant that continuously moves replicas from host to host, effectively keeping address and port information dynamic and hard to determine. We have extended this notion to make other aspects of the system, including its adaptive responses, dynamic and hard to predict. Further examples include port and address hopping,

unpredictable server selection for client-server interactions, and unpredictable network route selection for connections.

Local Tactics

The strategies we've described use a combination of defensive tactics and mechanisms linked with coordination glue code. *Tactics* are reactive defenses that use the capabilities of a small number of mechanisms. Typically, a tactic combines a sensor with an actuator mechanism to adapt to local situational changes.

Many local tactics are highly reusable and self-contained. Although their effectiveness in prolonging an application's useful life might be limited in isolation, local tactics are effective as building blocks of larger, overarching strategies.

We have implemented numerous tactics as part of the APOD toolkit:

- *Choking TCP connection floods.* This tactic uses Netstat, a widely available networking utility, in combination with a threshold counter to sense TCP connection floods and responds by blocking traffic from the flood's suspected source. This defense tactic responds quickly, but it can produce a relatively high false-positive rate in the presence of spoofed packets.
- *Blocking suspicious traffic.* This tactic combines the Snort network IDS (www.snort.org) with the Iptables Linux firewall (www.netfilter.org) to block traffic to and from a machine. Snort watches over network traffic for attacks and has a large database of well-known attack signatures.
- *Containing ARP cache poisoning.* The Address Resolution Protocol (ARP) spoof tactic continuously monitors the mapping of media-access control (MAC) to IP addresses. Upon detecting an attack, the tactic resets the cache with the correct values and blocks traffic to and from the offending MAC addresses.
- *Squelching insider floods.* This tactic monitors traffic flow and adaptively activates rate limiting upon observing packet floods. Running on a boundary controller, the tactic continuously monitors outgoing traffic via Iptables and calculates throughput metrics such as packets per second and bits per second. It then compares mean values for observed and expected parameters at regular intervals; if the observed outgoing traffic is significantly higher than expected, the tactic changes routing configurations at the boundary controllers to limit outgoing traffic.

- *Restoring corrupted files.* Upon sensing integrity violations of critical files via a file checker (such as Tripwire,¹¹ www.tripwire.com), this tactic uses the host controller to restore corrupted files from a noncorruptible backup medium (such as a CD-ROM).
- *Shutting down hosts.* However hard a defensive system tries to keep an attacker out, the attacker might still find some way to take over one of the hosts and use it as a launching point for further attacks. In such cases, the adaptive defense could fall back on the tactic of halting suspected hosts. However, blindly shutting down hosts based on local IDS information can leave the system vulnerable to attacks that cause self-inflicted denial of service. Therefore, we shouldn't deploy the host-shutdown tactic in isolation but rather as an integrated part of an attack-containment strategy.
- *Network IDSs.* The lightweight Snort signature-based IDS tries to identify network attacks by matching traffic to a set of attack patterns. The Internet community continuously updates the pattern-rule set as new attacks evolve.
- *Firewalls.* Iptables (also called Netfilter) implements a stateful firewall on top of Linux. Using Iptables, we 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.
- *TCP stack probes.* Netstat reports information about the TCP stack's internal state, including the number of parallel connections to a specific TCP port. In addition, the defense can obtain information about the MAC-to-IP address mapping by inspecting local ARP caches via the `arp` command.¹²
- *Virtual private networks.* VPNs let end systems establish trusted communication channels over untrusted networks such as the Internet by employing encryption and authentication mechanisms. The APOD toolkit has interfaces to FreeS/WAN (www.freeswan.org), a Linux implementation of IPsec, which lets applications dynamically establish tunnels. We also integrated lightweight user-space tunneling tools, including OpenSSH and Zebedee (www.winton.org.uk/zebedee).
- *Bandwidth-reservation and priority schemes.* Integrated Services (IntServ), an Internet Engineering Task Force (IETF) standard for network QoS based on reservations, as represented by RSVP, employs a signaling protocol to ensure end-to-end reservation attributes, requiring routers along the path to keep internal reservation state. Differentiated Services (DiffServ), in contrast, pushes the complexity to the network edges: traffic marking and conditioning takes place in border routers, letting the core routers treat packets according to their markings and eliminating the need to keep reservation state. In APOD, we experimented with two different RSVP implementations to defend the system against outsider floods: a Corba-wrapped version of the implementation from the University of Southern California Information Sciences Institute and a security-enhanced version of the University of Darmstadt implementation called Security Enhanced RSVP (SERSVP).¹³ In addition, we've implemented a simple bandwidth broker that interfaces to Linux and Cisco routers to change queuing policies.

Not all defensive tactics involve reactive adaptation. For example, *Port and address hopping* is a dynamic tactic that constantly changes a service's TCP identity – that is, its IP address and TCP port. The process is completely transparent to the application's end users. The intention is to both hide the service's real identity and confuse the attacker during reconnaissance. As an additional benefit, this tactic increases the likelihood of detecting an attack. The scope, however, moves beyond that of local tactics because the hopping service and its clients need to be coordinated so that the clients use the currently active IP address and port of the server. When dealing with applications protected by a secured perimeter, such as a demilitarized zone (DMZ), security engineers can configure this tactic to interface with firewalls to dynamically adapt the policy rules.

Mechanisms

Defense mechanisms provide the actions we can invoke as part of adaptive tactics and strategies. Each mechanism provides a certain detection, prevention, or recovery functionality. For example, sensors such as the Tripwire file checker and Snort network IDS look for suspicious activities. Actuators such as the Iptables firewall respond to attacks, and resource management mechanisms provide services such as data dissemination and component replication.

With the APOD toolkit, we integrated the following set of mechanisms:

- *Traffic shapers.* The Iproute2¹² package lets Linux hosts function as replacements for hardware routers. Linux routers provide full-fledged queue-management support, which makes both IntServ and DiffServ possible. We implemented traffic shaping through a token bucket filter queue. The defense can specify a maximum forwarding rate and further refine the policy to allow short-term bursts.
- *Process and data replicators.* The Self-Stabilizing Software Bus is a BBN-developed mechanism for data publication and process management. Given the information required to start a process, the bus attempts to start and maintain a specified number of copies of the process. A process that isn't started by the bus can also "attach" itself to the bus and ask for a handle to an object that the bus maintains. Any process attached to the bus can "put" data on the bus, which pushes the bus into an unstable state. As part of its self-stabilization algorithm, the bus then disseminates the new data to all bus processes, eventually restabilizing in the sense that all bus nodes agree on the bus data. The bus mechanism also corrects bus-process crashes and data corruption as it stabilizes. We have successfully used the bus to disseminate information between multiple nodes and replicate stateful application objects.
- *File-integrity checkers.* Tripwire computes cryptographic hashes for files and periodically checks whether they've been modified.
- *Host controllers.* We have developed a controller component that can securely execute selected Unix commands using the `sudo` Unix command. This mechanism is commonly used to replace files in the filesystem, reboot, and halt nodes.

All these mechanisms, the QuO adaptive middleware, and code for strategies and tactics are freely available as open-source software.

Decomposing and Aggregating Strategies

Tactics and mechanisms focusing on different aspects of the system must coordinate to successfully work together. Snort could misinterpret port hopping's frequent changes to packet destination ports as a port scan, for example, in the absence of coordination between the two mechanisms. Sophisticated defense strategies thus involve coordination among constituent (sub)strategies.

Figure 1 illustrates how the APOD toolkit lets us combine individual mechanisms and tactics into higher-level defense strategies. The overall strategy "Slow down an attacker through adaptive responses" is built on the attack-containment substrategy, which in turn uses the blocking-suspicious-traffic tactic implemented on top of the Snort and Iptables mechanisms.

In addition, the attack-containment substrategy uses the shutting-down-hosts tactic via components that run on all application hosts and coordinate their actions using the self-stabilizing software bus.

Upon detecting a port attack on host *X*, the attack-containment component checks whether the system is undergoing denial-of-service symptoms or experiencing a common mode failure by checking how many hosts have recently been infiltrated by the same attack. In the absence of denial-of-service attacks, it notes the source of the attack packet and disseminates the information to all other related components. Upon receiving the attack notification, the component on host *X* tries to initiate an automatic host shutdown to prevent the infiltrated system from being used as a launch point for further attacks. In addition, all other components block traffic to and from *X*.

If not carefully applied, traffic blocking can lead to self-inflicted denial of service through spoofed noisy attack packets. Therefore, the system applies containment only to a certain percentage of machines. After exceeding the limit, the attack-containment component issues recommendations only to the outrunning substrategy, which in turn avoids starting new replicas on infiltrated hosts, given that better choices are available.

To contain insider floods as part of attack containment, we deploy the squelch-insider-floods tactic on boundary controllers on each LAN. Because this defense only protects against floods originating from a LAN protected by a boundary controller, we also use SERSVP to defend against external floods of inter-LAN links. Finally, the boundary controllers are linked with host defenses to implement a traceback containment policy. If a boundary controller is notified that a host *H* within its domain is marked as suspicious, it blocks traffic to and from *H*, thus stopping floods before they cause problems.

Experimental Evaluation

As our adaptive defense capabilities matured, we conducted various experiments to test how effec-

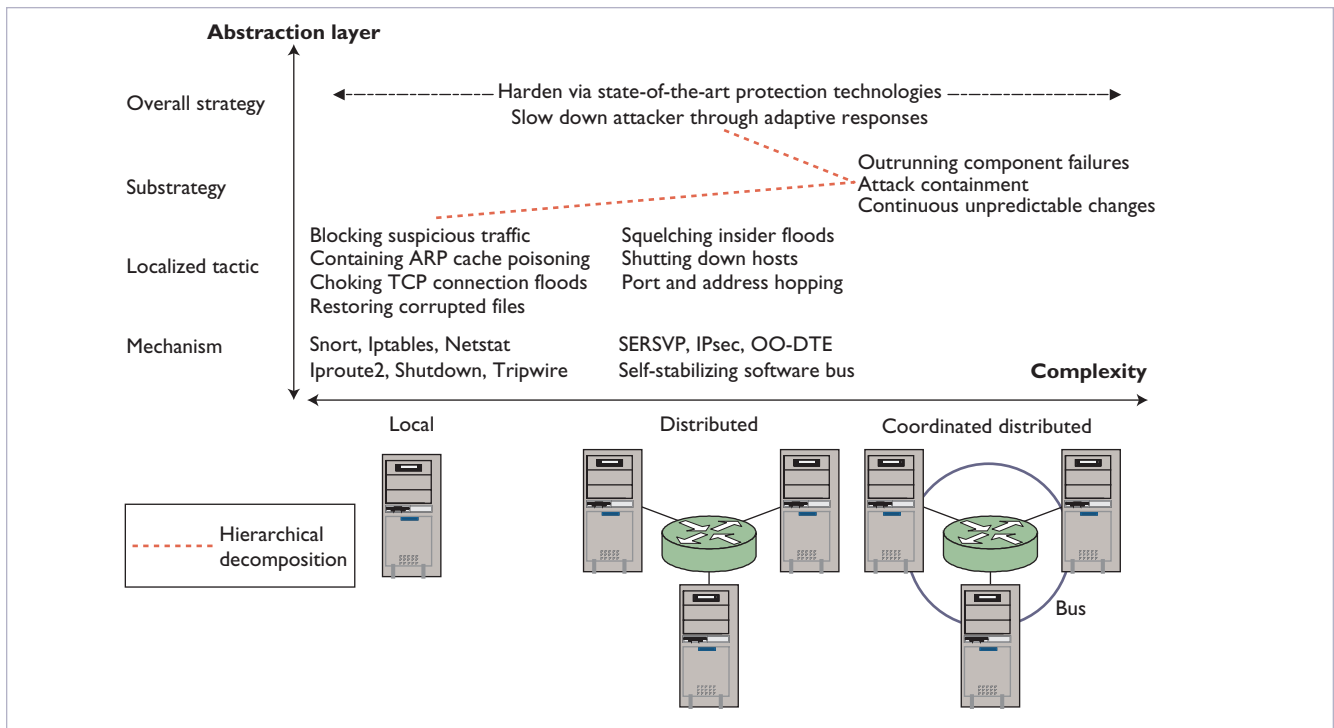


Figure 1. Integrating mechanisms and local tactics for an overall defense. In this sample of defense mechanisms, tactics, and strategies, the red line highlights an example decomposition of the overall defense.

tively they defended applications. Evaluations took place in multiple stages throughout the project. Initially, the developers used logical arguments to evaluate a given defense. Next, we subjected rapidly prototyped parts of the strategy to internal red-team tests. Finally, we participated in two formal red-team experiments under the umbrella of the DARPA Fault Tolerant Networks (FTN) and Dynamic Coalitions (DC) continuous experimentation program.¹⁴

The external red-team experiments aimed to investigate and quantify how well defense-enabling provides dynamic automated defense of a distributed application in a realistic network environment. The experimental application is representative of many client-server applications that must share data between multiple parties in a timely manner. Both clients and servers publish their information needs and information availability to an information broker, which connects clients to servers. Clients then directly request the information they need from the servers. Each instance of these three component types runs on its own host, distributed over a representative internetwork.

The first experiment (APOD-1) investigated how defense-enabling can help the application survive attacks that damage the broker component. Defens-

es in APOD-1 used replication in combination with the host-containment strategy and Snort and Iptables in a coordinated version of the block-suspicious-traffic tactic. The second experiment (APOD-2) expanded the investigation to include attacks that attempt to disrupt the information exchange in the application by flooding network links between routers. To address this, we incorporated a defense against TCP connection floods and SERSVP reservations and added squelching-insider-floods for tolerating flooding of network links.

The initial defense-enabling process (for APOD-1) took approximately four staff months, including the development of the experimental application, devising its survivability requirement, formulating its survivability strategy, and integrating with defense mechanisms. The next enhancement (for APOD-2) required another staff month to complete. In both cases, we had to develop only a minor fraction of the defense mechanisms from scratch, leveraging existing mechanisms that were already part of the toolkit.

The red team, an independent team separate from the APOD defense developers and selected by DARPA, used multistaged attacks, composed of subattacks and executed in a coordinated way. Each subattack was aimed at achieving a partial

goal toward bringing down the application. The red team started on an attacker host attached to the network, so obtaining a toehold in the system involved determining which hosts participate in the application and intruding on one of them. The coordination between subattacks required timing; in some cases, the defense completely withstood the attack by mounting adaptive responses to the attack's effects.

Although the red team eventually succeeded most of the time, the experiments demonstrated that integrating adaptive defenses with the subject application improved its immunity to sustained attack for nontrivial time periods. APOD-1 focused more on measuring the latency of requests being serviced because the experiment referees — another independent team separate from the red team and the APOD defense developers — expected the application to degrade as a result of attacks on the broker (clients wait until they find a server, hence the expected increase in latency). However, the adaptive defense strategy of moving the broker component away from affected hosts showed practically no latency degradation until the point of denial — that is, until the attackers killed the last running instance of the broker. In APOD-2, we measured time to denial and found that the red team required 45 minutes, on average, to break the defended application with a live attack. A scripted attack took on average 19 minutes, whereas undefended applications broke down within seconds. Further analysis showed that the APOD-based defense had a low false-positive rate — that is, responses weren't mounted spuriously when there were no attacks — and adapts quickly once an attack effect is observed.

The red team also attempted to exploit adaptive responses against the defense. One such response was to sacrifice a host from which suspicious (signature-based) traffic was detected. After intruding on a host, the red team was able to observe which hosts were key to the application and systematically spoof suspicious traffic so that it appeared to emanate from all these hosts. The defense strategy's threshold scheme, designed to avoid such self-inflicted denial of service, foiled this attack.

Overall, the experiments provided a positive reinforcement for the value and affordability of adaptive cyberdefense in the context of distributed applications. Detailed descriptions of the experiments, more experimental data, and in-depth analyses are available elsewhere.^{15–17}

Future Work

The APOD toolkit provides readily available interfaces to various mechanisms that we can use to defend against cyberattacks. This provides a wide base to build and experiment with higher-level defense strategies and their various combinations. We are continuing to investigate how to introduce more powerful and flexible adaptive response capabilities, and how to improve coordination among multiple distributed defense mechanisms. We are expanding our work with new ideas in adaptive distributed trust management, as well as making adaptive distributed capabilities more reusable and robust without adding additional vulnerabilities. □

Acknowledgments

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

References

1. P. Pal et al., "Defense Enabling Using Advanced Middleware: An Example," *Proc. Military Comm. Conf. (MILCOM)*, vol. 1, IEEE Press, 2001, pp. 92–101.
2. F. Webber et al., "Defense-Enabled Applications," *Proc. DARPA Information Survivability Conf. (DISCEX II)*, vol. 2, IEEE CS Press, 2001, pp. 119–125.
3. M. Atighetchi et al., "Adaptive Use of Network-Centric Mechanisms in Cyber-Defense," *Proc. 6th IEEE Int'l Symp. Object-Oriented Real-Time Distributed Computing*, IEEE CS Press, 2003, pp. 183–192.
4. M. Atighetchi et al., "Building Auto-Adaptive Distributed Applications: The QuO-APOD Experience," *Proc. 3rd Int'l Workshop Distributed Auto-adaptive and Reconfigurable Systems (DARES)*, IEEE CS Press, 2003, pp. 104–109.
5. D. Siewiorek and R. Swarz, *The Theory and Practice of Reliable System Design*, Digital Press, 1982.
6. M. Cukier et al., "Providing Intrusion Tolerance with ITUA," *Supplement of the 2002 Int'l Conf. Dependable Systems and Networks (DSN)*, IEEE Press, 2002, pp. C-5-1–C-5-3.
7. R. E. Schantz et al., "Packaging Quality of Service Control Behaviors for Reuse," *Proc. 2002 IEEE Int'l Symp. Object-Oriented Real-Time Distributed Computing (ISORC)*, IEEE Press, 2002, pp. 375–385.
8. J. Zinky et al., "Runtime Performance Modeling and Measurement of Adaptive Distributed Object Applications," *Proc. Int'l Symp. Distributed Object and Applications (DOA)*, LNCS 2519, Springer-Verlag, 2002, pp. 75–77.
9. J. P. Loyall et al., "Emerging Patterns in Adaptive, Distributed Real-Time, Embedded Middleware," *Proc. OOPSLA 2002 Workshop: Patterns in Distributed Real-Time and Embedded Systems*, 2002; <http://quo.bbn.com>.
10. G. Duzan et al., "Building Adaptive Distributed Applica-

tions with Middleware and Aspects," *Proc. Int'l Conf. Aspect-Oriented Software Development (AOSD)*, ACM Press, 2004, pp. 66–73.

11. G. Kim and E. Spafford, "The Design and Implementation of Tripwire: A Filesystem Integrity Checker," *Proc. 2nd ACM Conf. Computer and Comm. Security*, ACM Press, 1994, pp. 18–29.
12. B. Hubert et al., "Linux Advanced Routing and Traffic Control How-to," 2003, <http://ds9a.nl/2.4Networking/>.
13. T.-L. Wu et al., "Securing QoS: Threats to RSVP Messages and Their Countermeasures," *Proc. 7th Int'l Workshop on Quality of Service*, IEEE Press, 1999, pp. 62–64.
14. J. Lowry and K. Theriault, "Experimentation in the IA Program," *Proc DARPA Information Survivability Conf. (DISCEX II)*, vol. 1, IEEE Press, 2001, pp. 134–140.
15. B. Nelson et al., *APOD Experiment 1: Final Report*, tech. report 1311, BBN Technologies, May 2002; <http://apod.bbn.com>.
16. B. Nelson et al., *APOD Experiment 2: Final Report*, tech. report 1326, BBN Technologies, Sept. 2002; <http://apod.bbn.com>.
17. P. Pal et al., "Reflections on Evaluating Survivability: The APOD Experiments," *Proc. IEEE Int'l Symp. Network Computing and Applications (NCA 03)*, IEEE Press, 2003.

Michael Atighetchi is a senior scientist at BBN Technologies and a senior member of the Distributed Systems Advanced Middleware Technology group, where he conducts research on enabling technologies for advanced systems. His interests include use of adaptation in survivable systems, network and operating system security, and distributed coordination. Contact him at matighet@bbn.com.

Partha Pal is a division scientist at BBN Technologies, and he leads the survivability research thrust in the Distributed Systems Advanced Middleware Technology group. He is currently working on the architecture, design, implementation, and evaluation of a pathfinder system for the US Dept. of Defense. Pal is a senior member of the IEEE. Contact him at ppal@bbn.com.

Franklin Webber is an independent consultant, working as a full-time, senior member of the QuO Distributed Systems Advanced Middleware Technology group at BBN Technologies. His research interests include development and formal verification of software for secure and reliable systems. Contact him at fwebber@bbn.com.

Richard Schantz is a principal scientist at BBN Technologies and the technical director of the Distributed Systems Advanced Middleware Technology group. He has been involved in distributed systems research since its inception in the early 1970s. Contact him at schantz@bbn.com.

Christopher Jones works at TriGeo Network Security. He was a scientist at BBN Technologies and a member of the Distributed Systems Advanced Middleware Technology group until May 2004. While at BBN, he worked on using the services and capabilities of security tools in the applications defense by developing and integrating defense mechanisms in the QuO middleware. Contact him at cjones@trigeo.com.

Joseph Loyall is a division scientist at BBN Technologies, where he leads the Distributed Real-time Embedded (DRE) systems research thrust in the Distributed Systems Advanced Middleware Technology group. He is actively involved in developing integrated dynamic resource management capabilities and advanced software engineering using model-driven architecture (MDA) approaches, and in applying adaptive behavior to operational embedded systems such as collections of unmanned and manned air vehicles. Contact him at jloyall@bbn.com.

**DON'T
RUN THE RISK.
BE SECURE.**

IEEE
SECURITY & PRIVACY

Ensure that your networks operate safely and provide critical services even in the face of attacks. Develop lasting security solutions, with this peer-reviewed publication.

Top security professionals in the field share information you can rely on:

Wireless Security • Securing the Enterprise • Designing for Security Infrastructure Security • Privacy Issues • Legal Issues • Cybercrime • Digital Rights Management • Intellectual Property Protection and Piracy • The Security Profession • Education

Order your subscription today.

www.computer.org/security/

IEEE
IEEE COMPUTER SOCIETY
www.computer.org