Building Auto-Adaptive Distributed Applications: The QuO-APOD Experience *

Michael Atighetchi, Partha P. Pal, Christopher C. Jones, Paul Rubel,
Richard E. Schantz, Joseph P. Loyall, John A. Zinky
BBN Technologies LLC
{matighet,ppal,ccjones, prubel,schantz,jloyall,jzinky}@bbn.com

Abstract

Exploiting autonomic adaptation in defending a distributed application is a relatively new research area. We describe how the QuO adaptive middleware was used to implement auto-adaptive defenses ranging from simple rapid-response sensor-actuator tactics to more sophisticated containment and outrun strategies. In addition, we report on two experiments where live red team attacks were used to evaluate our auto-adaptive defense technology.

1 Introduction

The availability and quality of resources and services an application relies upon may vary over time in a distributed setting. Such variations may happen naturally, or they may be caused by a malicious attacker. In the most benign case, some design assumptions may be found to be improper at deployment time, or may become violated for a brief period during run time. More serious cases involve resource corruption/depletion attacks that threaten the very survival of the application. Therefore, survivability is clearly an area where advances in autonomic adaptation can provide a high pay-off.

We have been researching the issues underlying adaptive distributed systems for the last few years and have developed a general purpose adaptive middleware called QuO (Quality Objects) based on the distributed object computing (DOC) paradigm. Although the initial idea behind QuO was to support QoS and QoS-based adaptation within the context of distributed applications[20], QuO has evolved to a general purpose advanced middleware for auto-adaptive distributed systems. The QuO technology suite includes specification languages[13] and code generation tools to define adaptive behavior and a runtime system[16] to support it. Over the years, we have used QuO to provide adaptation to meet real time constraints[2] and to provide defensive responses against cyber attacks[9] in addition to more traditional QoS oriented adaptation. We have captured various adaptive middleware patterns[7] and have also used aspect oriented techniques to introduce adaptive behavior to existing applications[5]. In addition to numerous technology demonstrations, we performed measurement and assessment experiments to evaluate the effectiveness of our adaptive capabilities[12]. We have also been engaged in various successful technology transfer efforts where more mature components of our technology are integrated into advanced military systems.

With a relatively better understanding of our current capability and its utility, we are continuing to investigate further enhancements as well as the theoretical underpinnings of adaptive distributed systems. Our position on auto-adaptive distributed systems can be summarized as follows. First, we take a middleware-centric approach to build auto-adaptive behavior into distributed systems. Middleware offers the ability to monitor and control both the application and the infrastructure, and makes it possible to introduce the adaptive behavior with minimal changes. Second, we believe that auto-adaptive behavior is crucial for survivability and in this context auto-adaptive behavior must integrate and coordinate the services and capabilities of multiple mechanisms that are normally part of the original application. Third, we believe that a systematic, modular and reusable way to build and integrate auto-adaptive behavior is necessary for making auto-adaptive capabilities useful in a practical way. Finally, we view that a careful specification and evaluation of the claims made by auto-adaptive mechanism is a prerequisite for its wide acceptance in real life applications.

In this paper we present an abbreviated view of our experience in developing and evaluating auto-adaptive distributed applications. We will focus on the use of auto-adaptive capabilities in terms of defending against cyber attacks, and in particular we will present a narrow sample of adaptive responses that make use of network-based

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

tools and mechanisms. The rest of the paper is organized as follows. Section 2 describes the QuO adaptive middleware in short. In the APOD project we investigated the use and effectiveness of auto-adaptive capabilities in defending the application against attackers, a process we call *defense enabling*. Section 3 describes the network oriented adaptive responses that we developed in this project. Section 4 describes various experiments we carried out in order to evaluate the effectiveness of such adaptive behavior in defense. Section 5 concludes the paper.

2 QuO Adaptive Middleware

QuO is a middleware framework for building applications that are aware of their environment and can adapt automatically to changes in it. QuO applications can specify their non-functional requirements (e.g., security, performance, or dependability requirements), measure what is being provided, access interfaces for controlling the desired level of service , and adapt to changes in the environment.

QuO has been used extensively in defense enabling because of its support for adaptation and its capability to integrate various mechanisms together. Building defense in middleware separates the specification and implementation of the defense from the functional aspects of an application. This separation facilitates the reuse of parts of the defense across multiple applications.

The QuO middleware framework provides a set of high-level languages known as QDLs (Quality Description Languages) as well as the runtime components needed to support them. QDLs take the same level of abstraction as Interface Description Languages in the DOC paradigm. QDL allows systematic specification of both an application's QoS requirements and the adaptive responses to take when these requirements are not met. Although designed initially to capture traditional application level QoS aspects such as response time, number of frames in a video transmission, etc., QDL can be used to capture any non-functional requirement that is important for an application. QDL makes auto-adaptive systems easier to create and allows the designers to focus on getting how and when to adapt while relieving themselves of functional issues while they do so.

3 Auto-Adaptive Network Defenses in APOD

This section describes a sample of the auto-adaptive responses that were implemented in the APOD (Applications that Participate in their Own Defense) project. The focus is mostly on behavior that observes essential network parameters, controls network resources to yield a better defense, and reacts to network related problems such as bandwidth exhaustion through flooding or TCP stack attacks.

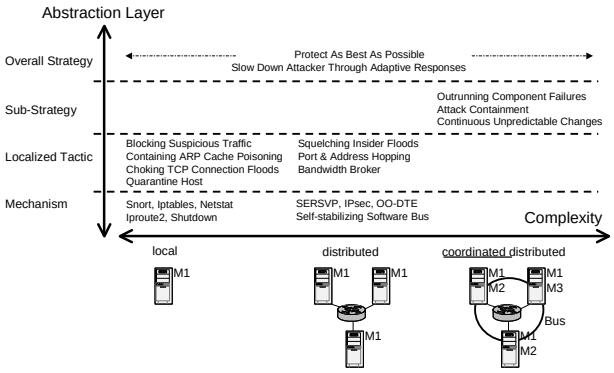

Figure 1. Integration of mechanisms and local tactics in an overall defense

We first describe the mechanisms that offer network-based capabilities used in defense. Next, local tactics are described that provide fast reactions based on local knowledge. Coordination of multiple tactics via a defense strategy concludes this section.

3.1 Defense Mechanisms

We use the following adjustable defense mechanisms as a basis for our adaptation:

- **Intrusion Detection** - Snort [14]
- **Firewalls** - Netfilter/Iptables [10]
- **TCP Stack Probes** - Netstat [15]
- **Virtual Private Networks** - FreeS/WAN [1],IPsec [4], openssh [11], and Zebedee [19]
- **Bandwidth Reservation Schemes** - RSVP [13], Security Enhanced RSVP (SERSVP)[18]
- **Traffic Shaping** - Iproute2 [3]

Each mechanism performs a specific security related task: Snort provides attack alerts, Iptables can block traffic, Iproute2 rate limits traffic.

3.2 Adaptive Response as Defense Tactics

A simple use of network-centric defense mechanisms involves reactive responses with fairly local scope that utilize the capabilities of a small number of mechanisms. Such tactics tend to use one mechanism as a sensor and use a second mechanism for reaction. Figure 1 summarizes a sampling of the tactics we have investigated in APOD.

Many local tactics are highly reusable and self-contained. Although their effectiveness in prolonging an

application's useful life may be limited in isolation, local tactics are quite effective as components of larger, overarching strategies.

The current set of tactics consists of the 4 different defense behaviors that adapt due to attacks:

- *Choking TCP Connection Floods* - This tactic, used against TCP connection floods, uses Netstat in combination with a threshold counter to sense TCP connection floods. It then responds 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 mislabeled or spoofed messages.

- *Blocking Suspicious Traffic* - This tactic combines the Snort network intrusion detection system with the Iptables Linux firewall to block traffic to and from a machine.

- *Containing ARP Cache Poisoning* - The ARP spoof defense continuously monitors the mapping of MAC to IP addresses. Upon detecting an attack, the cache is reset with the correct values and traffic to and from offending MAC addresses is blocked.

- *Squelching Insider Floods* - This defense mechanism stipulates the notion of boundary controllers to contain floods. The flood containment tactic (running on a boundary controller) continuously monitors outgoing traffic via Iptables and calculates throughput metrics such as 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 at 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.

Not all defensive tactics need to involve reactive adaptation. A case in point is a defense mechanism called *Port and Address Hopping*. 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. An additional benefit of this tactic is that it increases the likelihood of detecting an attacker. Note that the scope of this tactic is not local anymore since the hopping service and its consumers will need some kind of coordination.

3.3 Defense Strategies

The individual adaptive responses described as part of local tactics are not enough to effectively defend critical applications. Multiple tactics are often required to address an application's survivability requirements. Figure 1 illustrates how APOD allows one to combine individual mechanisms and tactics into higher-level defense strategies.

Tactics and mechanisms focusing on different aspects of the system need to coordinate in order 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 the absence of coordination between the two mechanisms. Therefore, more sophisticated defense strategies involve coordination among constituent (sub-)strategies to be built into 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. This hardening forms the first line of defense. Defense enabling builds more dynamic responses on top of a hardened foundation, assuming the availability of many key network technologies, including IPsec VPNs, ssh tunnels, firewalls, and RSVP reservations.

The defense strategy aims to significantly improve the first line of defense by managing the shortcomings and failures of the protection it provides. There are three aspects of this strategy that can be termed as sub-strategies. They are:

- *Outrunning Component Failures*: Replicates key application components and intelligently places new replicas on a suitable hosts upon noticing component failures.

- *Attack Containment*: Isolates host intrusions and network-based distributed denial of service attacks and stops their propagation.

- *Continuous Unpredictable Changes*: Places strict time constraints on the usefulness of obtained attack information by constantly changing unpredictably.

We explain the auto-adaptive features of these sub-strategies in more detail for the remainder of this section.

3.3.1 Outrunning Component Failures

Replicating components allows the system to continue even if some of the replicas fail. Consumers of data from the replicated components are unaffected by the loss of a replica. We can mask component failures by restarting new replicas to replace the ones that have failed . The outrun

sub-strategy restarts replicas on hosts believed to be more secure and therefore keeps them away from attackers.

Within APOD we use a self-stabilizing software bus ("bus" for short) to tolerate crash failures. The outrun sub-strategy interfaces with the bus and, upon detecting a replica death, selects a new host for the replacement replica. The host is picked from a list of possible candidates by searching for a host which is hard for the attacker to infiltrate based on the defense's current knowledge of the attack state. The sub-strategy gives higher preference to hosts located in a different security domain, i.e., on a different IP subnet, relative to the observed fault. Further network centric information, such as whether a network level intrusion has been observed on the candidate or its network, is also considered.

3.3.2 Attack Containment

This sub-strategy uses a combination of tactics to prevent an attacker from being able to attack large parts of the system by taking over a single part of it. This directly relates to the notion of security domains to split the system into independent privilege realms. In the *Quarantine Hosts* tactic, individual machines are treated as security domains, and containment involves shutting down a machine upon detecting an intrusion. This behavior prevents an attacker who gains root privileges on one essential server from easily spreading the same attack to remaining servers.

Components implementing this tactic run on all application hosts and coordinate their actions using the bus. The components implement the *Block Suspicious Traffic* tactic combined with coordination logic to consistently block attacks and prevent self-inflicted denial of service.

Upon detecting a port attack on host X, the *Attack Containment* component checks whether the system 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 interested 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 using as a launch pad for further attacks. In addition, all other components block traffic to and from X. To prevent self-inflicted denial of service, the components only contain up to a certain percentage of machines. The exact number can be picked randomly during runtime. After exceeding the limit, the *Attack Containment* component only issues recommendations to the outrunning sub-strategy which in turn avoids starting new replicas on infiltrated hosts, given that better choices are available.

To contain insider floods, the *Squelch Insider Floods* tactic is deployed on boundary controllers on each LAN. Since this defense only protects against floods originating from a LAN protected by a boundary controller, SERSVP is used in addition to defend against external floods of inter-LAN links. 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.

3.3.3 Continuous Unpredictable Changes

Sophisticated attackers often spend a large amount of effort gathering information about essential services. Under the assumption that determined attackers will get any information given enough time the defense sub-strategy described here tries to render information obtained by attackers harmless by changing it frequently. This has the added benefit that the use of stale information can be used to raise alerts to intrusion detection systems.

This sub-strategy has much in common with the reactive version of the outrunning sub-strategy described above. A proactive outrunning sub-strategy would keep moving objects from host to host, effectively keeping address and port information dynamic and hard to determine. This sub-strategy takes this notion further and attempts to make other aspects of the system, including its adaptive responses dynamic.

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 sub-strategy has to be flexible enough to allow one to describe this trade-off upon deployment. Examples of this sub-strategy include port and address hopping, unpredictable server selection for client server interactions, and unpredictable network route selection for connections.

Although this sub-strategy is proactive, a reactive version may make sense if the defense can survive the attack at hand. As in replication, a system which relocates replicas constantly might have unnecessary overhead compared to a system which starts a new replica unpredictably in a clean domain only upon noticing a fault.

4 Experimental Evaluation

As APOD was developed we conducted various experiments to test whether it was effective in protecting an application. We used these experiments to see how the ideas of auto-adaptation and survivability held up under attack. This evaluation took place in multiple stages throughout the project and involved both internal testing as well as internal and external Red Teaming (testing by parties not involved in the development).

Initially, the APOD developers used logical argument construction techniques to derive a design of defense mechanisms and strategies. Next, we rapidly prototyped parts of the strategy and subjected them to internal Red Team testing by a non-APOD developer. This early experimentation yielded many useful insights into what is needed to stay survivable in the presence of real attacks. Towards the end of the APOD project, the FTN/DC continuous experimentation program[6, 17] conducted two formal Red Team experiments to evaluate the maturity and applicability of APOD's auto-adaptive middleware approach to dynamic defense. The next two sections examine the experiments carried out and the results we collected from them.

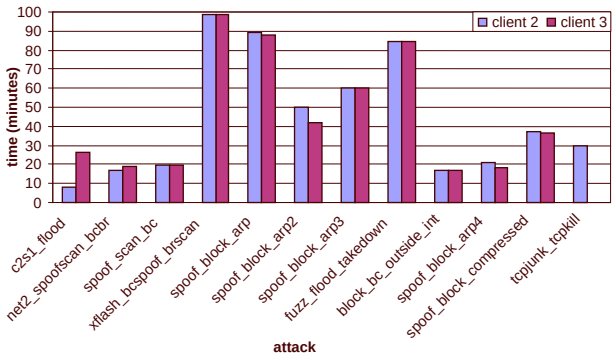

Figure 2. Time to Denial by Live Attack

4.1 Red Team Experiments

The overall goal of these experiments was to investigate and quantify the ability of defense enabling to provide dynamic automated defense of a representative application, an image-serving system, in a realistic, distributed environment.

The application used in the experiments is representative of many applications which 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 interact with the servers to get the desired information.

The first experiment (APOD-1) investigated how defense enabling can help the image serving application survive attacks that damage the broker component. The second experiment (APOD-2) expanded the investigation of APOD-1 to include attacks that attempt to disrupt the image serving application by flooding network links between routers.

The remainder of this section focuses on describing experimental results regarding network-level mechanisms and strategies. See [9, 8, 12] for further details about the experiments, including hypotheses, flags, rules of engagements, metrics, data analyses, and attacks.

4.2 Experiment Results

The APOD-1 experiment implemented replication in combination with the host containment strategy, utilizing Snort and Iptables in a coordinated version of the *Block Suspicious Traffic* tactic.

The Red Team attempted to exploit the defense against itself to capture the flags. For instance, one aspect of the overall defense strategy was to sacrifice a host from which suspicious traffic was detected. The Red Team attempted to misuse the system's auto-adaptation logic and have the system sacrifice all the hosts. This attack did not work because the auto-adaptive strategy used a threshold scheme to avoid such self-inflicted denial of service.

Successful attacks turned out to be 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. The coordination between sub-attacks required timing and in some cases the defense was able to completely withstand the attack. Despite the fact that the Red Team was able to eventually capture the denial flag most of the time, APOD-1 demonstrated that the auto-adaptive features, even in their early stages of development, integrated with the subject application improved its immunity to sustained attack for non-trivial time intervals at an acceptable cost.

Many scripted attacks relied on transient information such as TCP ports to work properly, which was disclosed to the Red Team for practical purposes. However, dynamic defenses to prevent attackers from obtaining such information in reality are an important aspect of the defense. The *Outrunning Component Failures* strategy was extended to accommodate a proactive mode in which replicas get constantly relocated to minimize the usefulness of information about their location obtained by attackers.

In APOD-2, the new defense against TCP connection floods was incorporated. SERSVP reservations and *Squelching Insider Floods* were added for tolerating flooding of network links.

SERSVP was successful at keeping bandwidth reservations up during most of the attacks. The added security provided by SERSVP was able to minimize effects of replay attacks on service degradation. The TCP connection flood defense was tested by the Red Team and was found to be effective. No outbound flooding attack was attempted by the Red Team, which left this part of the defense strategy untested.

The defended application held up well against a wide range of attacks in APOD-2 by detecting the attack or its effect on the system and mounting autonomic adaptive responses in defense. As seen in figure 2 on average the Red

Team required 45 minutes to break the defended application with a live attack. A scripted attack took on average 19 minutes where an undefended application would have succumbed almost immediately. Further analysis showed that APOD has a low false-positive rate and adapts quickly once an attack has been detected. More data and in depth analysis can be found in [9, 8, 12].

5 Conclusion

Our experience with the QuO adaptive middleware reinforces our emphasis on a middleware-centric approach to auto-adaptive distributed systems. Advanced middleware such as QuO can be used to provide structure to the specific adaptations used to provide survivability. Our experience in the APOD project also provides a positive indication that auto-adaptive behavior making use of multiple tools or mechanisms can be easily integrated with an application. The APOD experiments, in a limited way, showed that such adaptive behaviors can be used as a basis for enabling useful defensive responses to attacks. We were also able to encapsulate and deploy useful auto-adaptive behavior as middleware components. However, auto-adaptive distributed systems for survivability is a large research space, which we are continuing to explore. A representative sample of issues that need investigation includes a) introducing more powerful and more flexible auto-adaptive capabilities, b) managing distributed coordination, trust, and mutual interference among various auto-adaptive responses and c) making auto-adaptive distributed capabilities more reusable and robust without adding additional vulnerabilities.

References

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

[2] C. D. Gill, J. P. Loyall, R. E. Schantz, and D. C. Schmidt. Experiences using adaptive middleware in distributed real-time embedded application contexts: a dependability perspective. In *Workshop on Dependable Middleware-Based Systems, Part of Dependable Systems and Networks Conference (DSN 2002)*, June 2002. Bethedsa, MA.

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

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

[5] D. A. Karr, C. Rodrigues, J. P. Loyall, R. E. Schantz, Y. Krishnamurthy, I. Pyarali, and D. C. Schmidt. Application of the quo quality-of-service framework to a distributed video application. In *International Symposium on Distributed Objects and Applications*, Sept. 2001.

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

[7] J. P. Loyall, P. Rubel, M. Atighetchi, R. E. Schantz, and J. A. Zinky. Emerging patterns in adaptive, distributed real-time, embedded middleware. In *OOPSLA 2002 Workshop - Patternsin Distributed Real-Time and Embedded Systems*, Nov. 2002. Seattle, WA.

[8] 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.

[9] 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.

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

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

[12] 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.

[13] The quality objects (quo) users' and reference guide. http://quo.bbn.com.

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

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

[16] R. Vanegas et al. QuO's runtime support for quality of service in distributed objects. *Proc. Middleware 98, the IFIP Int'l Conf. on Distributed Systems Platform and Open Distributed Processing*, Sept. 1998.

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

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

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

[20] 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.