Generating Policies for Defense in Depth (DiD)

by Paul Rubel, Charles Payne, Michael Ihde, Steven Harp, and Michael Atighetchi

In 2002, the Defense Advanced Research Projects Agency (DARPA) challenged the research community to design and demonstrate an unprecedented level of survivability for an existing US Department of Defense (DoD) information system by combining Commercial-Off-The-Shelf (COTS) technologies with those developed by DARPA. In particular, DARPA required that the undefended system—a large, distributed, Publish/Subscribe/Query (PSQ) system, implemented using the Joint Battlespace Infosphere (JBI)—be defense enabled in such a way as to survive 12 hours of sustained attack from a Red Team modeling a nation-state adversary. (See Figure 1 for a notional diagram.) The development team, led by BBN Technologies, produced a solution architecture entitled *Designing Protection and Adaptation into a Survivability Architecture* (DPASA) [1] that combines the following elements:

- **Protection**—the ability to detect or prevent attacks;
- **Detection**—the ability to detect and report attack-related events; and
- **Adaptation**—the ability to modify system behavior and structure to repair attack damage or to degrade gracefully. (See Figure 2.)

In this article we focus on the defense mechanisms and policies that protect the system's network communications.

From a protection perspective, DPASA's goal was to block an attacker using the Defense in Depth (DiD) strategy illustrated in Figure 3. (The defense layer is shown in boldface, while the prevention technology(s) used at that layer appears in italics.) At the system layer, redundant hosts were deployed so that the failure of a single host would not stop the entire system. At the network layer, hosts were grouped into enclaves, and enclave-to-enclave communication was restricted and encrypted using a Virtual Private Network (VPN) firewall and router. At the host layer, authorized host-to-host communication was enforced by the Autonomic Distributed Firewall (ADF), a host-based, embedded,

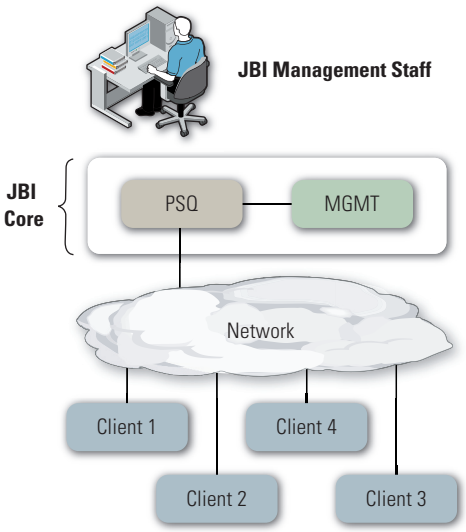
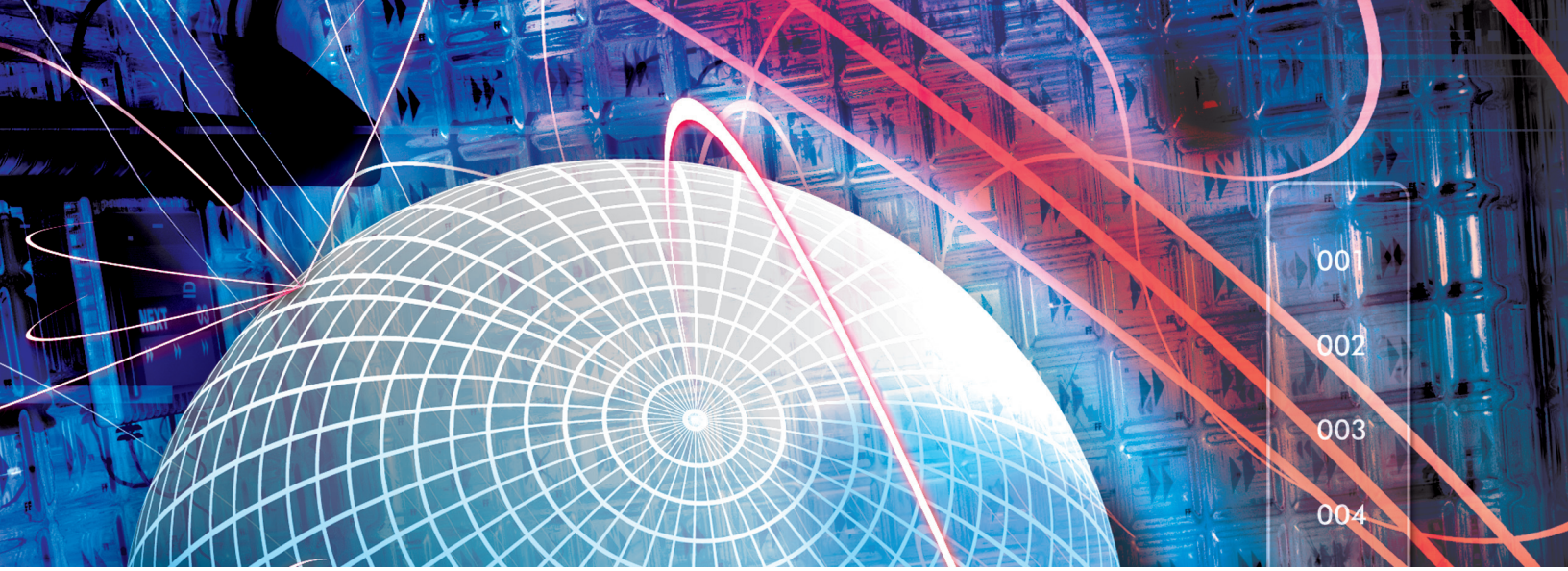


Figure 1 Baseline JBI

distributed firewall that is implemented on the host's Network Interface Card (NIC) and performs ingress and egress packet filtering. ADFs protect the host from the network and the network from the host. All host-to-host communication was also encrypted using ADF's Virtual Private Groups (VPG) [4], which provided a unique encryption key for each collection of hosts. At the process layer, authorized process behavior was enforced either by the National Security Agency's (NSA) Security Enhanced Linux (SELinux) or by Cisco System's Cisco Security Agent (CSA) for non-Linux hosts. At the application layer, the Java Virtual Machine (JVM) enforced authorized JBI application behavior. An application's network communications were subjected to defense mechanisms in each network, host, process, and application layer.

Constructing the policies that govern these defense mechanisms across the multiple layers in a coherent and mutually consistent way proved to be a challenge on several fronts. The richness of DPASA's DiD strategy meant there was significant *vertical* duplication of logical policy rules across the defense layers. Because of the nature of DiD, significant *horizontal* duplication also occurred between redundant hosts and network elements. For example, to minimize common mode failures, a mix of operating systems was used when providing redundant services. This meant that



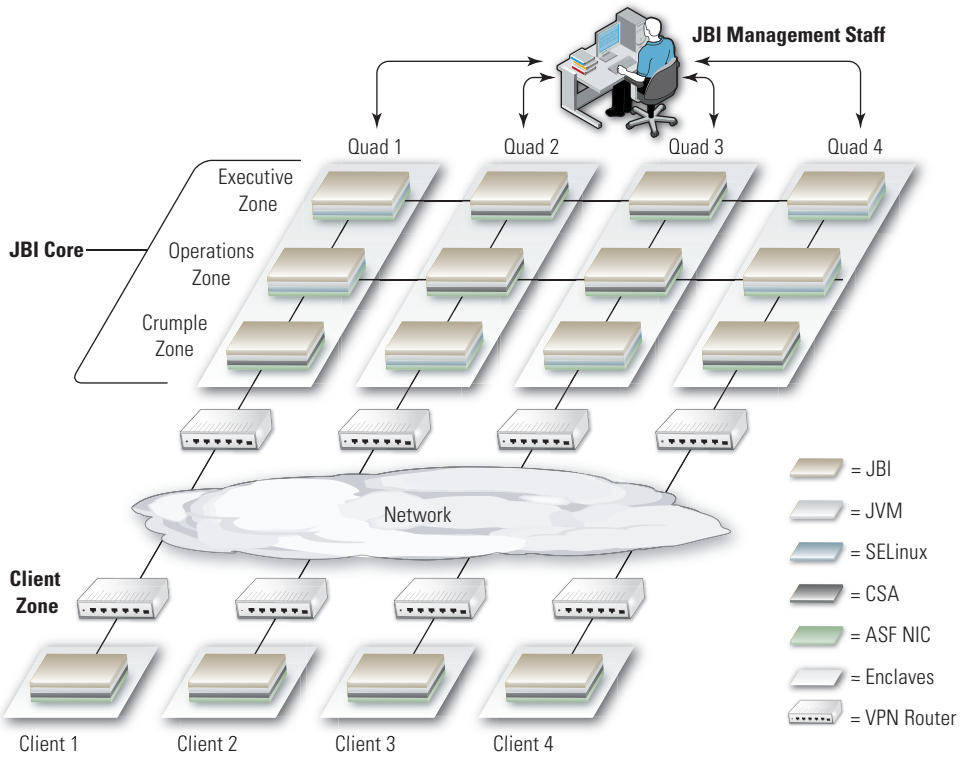


Figure 2 Survivable JBI

actual policies enforced by similar hosts could differ significantly (*e.g.*, between an SELinux host and a CSA-enabled host), even though those hosts were performing identical logical functions.

Strategy

There is a strong motivation to develop a single specification from which all policies will be derived. This topic has been the focus of significant research (*cf.*, [5], [6], [7]), which has demonstrated that a master specification can eliminate unnecessary duplication and be analyzed effectively for desired properties. However, those research efforts focused on policy coordination for identical or similar defenses within a single defense layer. What about policy coordination across multiple defense layers, as in a DiD system? The variety of enforcement targets and range of abstractions, from IP addresses and gateways to network services and processes, means that any useful master specification must contain many details at discordant levels of abstraction; *i.e.*, not all details are required at all defense layers, and unnecessary details tend to get in the way when reasoning about a layer at which they are not required. A master specification also raises concerns about hidden assumptions that might yield exploitable vulnerabilities and circumvent any gains promised by DiD by compromising all layers at once.

We initially pursued the master specification approach for selected layers. For example, we began by generating the host-layer policy automatically from the application-layer policy. However, constraints on policy construction at the host layer soon made this process unwieldy. We then moved to a hybrid approach that created policies in a coordinated but largely independent fashion. This yielded the best balance of flexibility, autonomy (an important quality for DiD), and assurance of correctness. The approach relied on a central specification of common values, such as host names and port numbers, plus policy templates; *e.g.*, for Java (which enforced application-level defenses), SELinux (which enforced process-level defenses), and ADF (which enforced host-level defenses), that relied on these values. Changes to these common values could then be propagated automatically to all affected policies. Where possible, we used existing policies

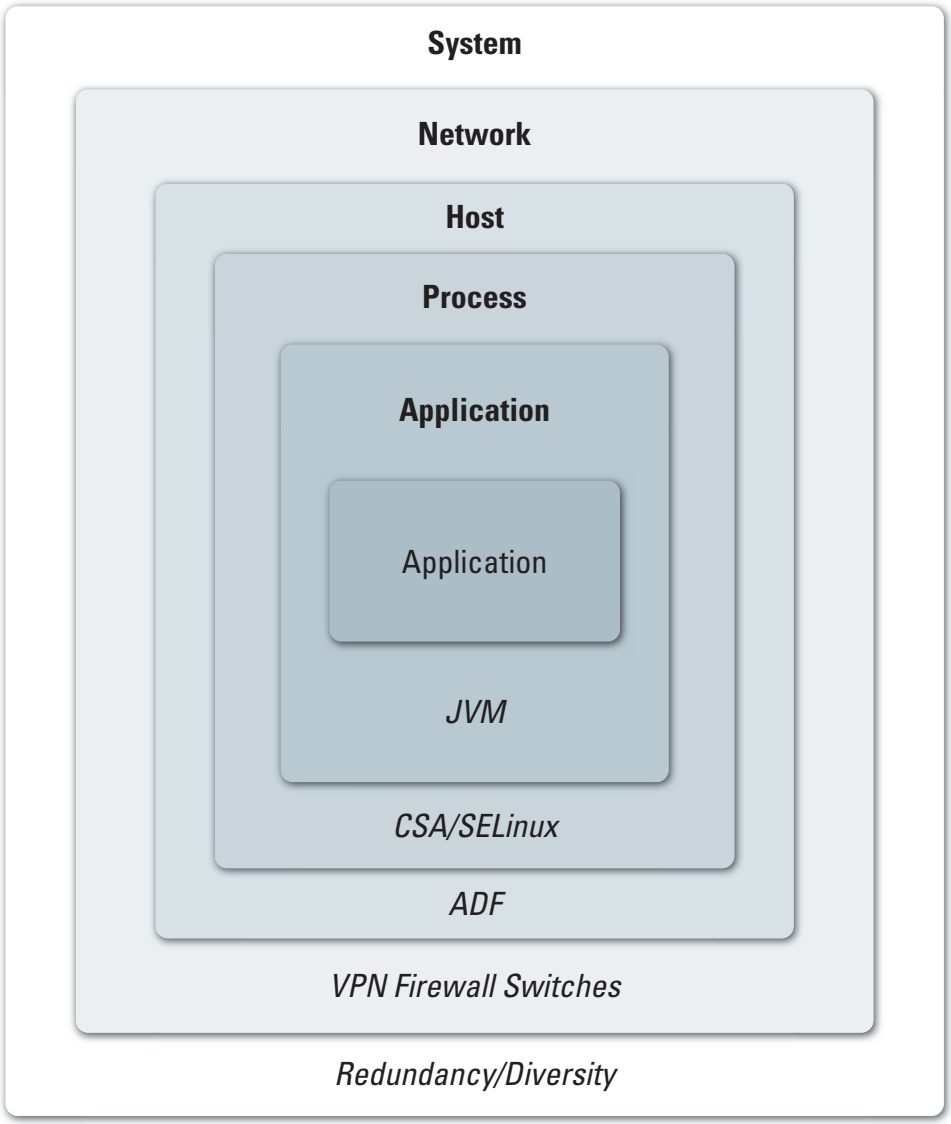


Figure 3 Attacker's Perspective of DPASA's Defense in Depth. (The defense layer is shown in boldface, while the prevention technology(s) used at that layer appears in italics.)

to inform new policies but did not automatically generate one from the other. For example, policies for CSA (which enforced process-level defenses) were developed independently from SELinux policies but were heavily informed by them.

To avoid simple misconfigurations, our hybrid approach shared information where appropriate but was not dogmatic about creating a single master specification. This approach also allowed us to use a single source to coordinate static policy elements shared by all policies. Separating functional roles from actual values allowed roles and values to change independently but kept then in synchronization between layers. This hybrid approach further minimized the risk of hidden assumptions by:

- Specifying each policy separately, using a different author,
- Structuring each policy to deny everything that is not explicitly allowed and then defining the policy according to observed failures to achieve a policy that was minimally sufficient, and
- Generating output to support manual and automated policy validation.

Generating validation support tools, such as visualizations and automated probing and testing, enabled software developers to review policies for correctness even if they did not understand the syntax of the policy-enforcement mechanism. For example, one automated validation task compared ADF policy against a thorough network scan. To conduct this task, we initiated a network scan from each host in the system to every other host, capturing the combined effects of egress filtering on the sending hosts and ingress filtering on the receiving hosts. The scan results were then automatically compared with the ADF policy to discover misconfigurations and unnecessary communication paths.

Lessons Learned

As in any large project, coordination and separation of concerns were important for making progress. In DPASA, we learned a number of valuable lessons along these lines as we developed and deployed the system.

- Our hybrid-policy construction worked well, because various authors could develop policies both independently and simultaneously. This was especially important, since policy authors were geographically dispersed. It also meant that it was not necessary for all authors to develop expertise in all technologies.
- Integration was greatly improved by having tools that supported command-line and file-based interactions. While Web-based interfaces are probably the friendliest for a novice user, they are awkward to integrate into a larger, multi-policy environment such as DPASA.
- Co-development of system functionality and policy is risky—developing policy against evolving system functionality causes undesirable churning. This can be avoided using one of two methods. First, by setting an acceptable but perhaps overly broad policy early on and then implementing the system to fit within the specified policy. This can be followed by a final refinement and tightening of the policy. Second, by implementing the system, being mindful of security concerns, and then creating the policy to tightly

fit the system requirements, as implemented. Since the undefended system did not have an existing set of policies, and we needed to incrementally build up the defense under a tight schedule, we chose the second option, which requires only one phase of policy construction.

- As the system grew, isolated testing became more difficult. Many policy refinements depended on observing the system in operation in a permissive mode, while collecting denial audits. In most cases, it was impossible to fully test applications in isolation, as related applications also had to be running. The constantly changing and challenging-to-test system impeded policy development to a surprising degree and underscored the need to reduce coupling between applications.

Conclusions

In DiD-enabled systems, constructing each policy in isolation is labor intensive and can lead to configuration errors. However, generating all policies from a single specification—an approach advocated for policies within a particular defense layer such as the network layer—is perhaps even more labor intensive and prone to error for DiD solutions, because too many details in that specification will apply only to specific layers, creating an unwieldy specification. Instead, we advocate a hybrid approach that:

- Encourages selective sharing of policy elements, while maintaining policy autonomy,
- Encourages independence between policy authors to reduce common, faulty assumptions,
- Builds policies from observed failures to be minimally sufficient, and
- Integrates validation tools to support other policy stakeholders.

Such an approach minimizes the risk of exploitable vulnerabilities that could circumvent the benefits of DiD.

A critical measure of success, of course, is how well the resulting policies and the defense mechanisms enforcing them perform against a determined adversary. At this writing, analysis of the Red Team assessment of DPASA is ongoing; however, preliminary results confirm that the overall DPASA solution architecture puts up a formidable defense. We believe that this approach is a solid step forward for making enforcement of multi-layer defense both cost-effective and practical. ■

Acknowledgement

This work was supported by DARPA under Contract number F30602-02-C-0134.

References

1. Atighetchi, Michael, Paul Rubel, Partha Pal, Jennifer Chong, & Lyle Sudin. Networking Aspects in the DPASA Survivability Architecture: an Experience Report. *Proceedings of The 4th IEEE International Symposium on Network Computing and Applications (IEEE NCA05)*, Cambridge, MA, 2005.
2. Chong, Jennifer, Partha Pal, Michael Atighetchi, Paul Rubel, & Franklin Webber. Survivability Architecture of a Mission Critical System: The DPASA example. *Proceedings of the 21st Annual Computer Security Applications Conference*, Tucson, AZ, 2005.
3. Rubel, Paul, and Michael Ihde, Steven Harp, & Charles Payne. Generating Policies for Defense in Depth. *Proceedings of the 21st Annual Computer Security Applications Conference*, Tucson, AZ, 2005.
4. Markham, Tom, Lynn Meredith, & Charles Payne. Distributed Embedded Firewalls with Virtual Private Groups. *DARPA Information Survivability Conference and Exposition Volume II*, 2003.
5. Bartal, Yair, Alain Mayer, Kobbi Nissim, & Avishai Wool. Firmato: A Novel Firewall Management Toolkit. *ACM Transactions on Computer Systems*, 22(4):381–420, November 2004.
6. Uribe, Tomas E., & Steven Cheung. Automatic Analysis of Firewall and Network Intrusion Detection System Configurations. *Proceedings of the 2004 ACM Workshop on Formal Methods in Security Engineering*, Washington, DC, 2004.
7. Guttman, Joshua D. Filtering Postures: Local Enforcement for Global Policies. *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, Oakland, CA, 1997.

About the Authors

Paul Rubel | is a staff scientist in the Quality Objects (QuO) group at BBN Technologies. His research focuses on adaptive distributed systems, specifically fault-tolerant middleware and survivable systems. Mr. Rubel is a member of the Institute of Electrical & Electronics Engineers (IEEE) and can be reached at prubel@bbn.com.

Charles Payne | has performed computer security research for nearly 20 years at industry and government laboratories. His work has focused on simplifying the definition and management of computer security policies and on streamlining the construction of convincing assurance arguments to ensure that these policies are upheld. Other research interests include host-based, distributed firewalls, high-assurance computing systems, and formal methods. Mr. Payne is a member of the IEEE Computer Society and may be reached at charles.payne@adventiumlabs.org.

Michael Ihde | received an MS from the University of Illinois and is currently a Software Engineer with Northrop Grumman Information Technology. As a graduate student, he conducted research on verification and validation of survivable, distributed systems, with an emphasis on distributed firewall technologies. He may be reached at mike.ihde@randomwalking.com.

Steven Harp, PhD | is a staff scientist at Adventium Labs with a background in cognitive science, artificial intelligence, and statistics. He has been studying applications of automated reasoning to problems in computer and physical security for the last five years. He may be reached at steven.harp@adventiumlabs.org.

Michael Atighetchi | is a Senior Scientist in the Quality Objects (QuO) group at BBN Technologies. He conducts research on enabling technologies for advanced distributed systems, with a major focus on using adaptation in survivable systems, network and operating system security, and distributed coordination. He may be contacted at matighet@bbn.com.

Footnotes

1. Designing Protection and Adaptation into a Survivability Architecture. More details about the architecture can be found in [1], [2], and [3].