

Supporting Safe Content-Inspection of Web Traffic¹

Dr. Partha Pal and Michael Atighetchi
BBN Technologies

Interception of software interaction for the purpose of introducing additional functionality or alternative behavior is a well-known software engineering technique that has been used successfully for various reasons, including security. Software wrappers, firewalls, Web proxies, and a number of middleware constructs all depend on interception to achieve their respective security, fault tolerance, interoperability, or load balancing objectives. Web proxies, as used by organizations to monitor and secure Web traffic into and out of their internal networks, provide another important example.

As more and more interactions (including personal, financial, and social) become Web-based, a number of observations can be made. First, as technology advances and public awareness of Internet security increases, an increasing portion of Web traffic is likely to be carried by Hypertext Transfer Protocol Secure (HTTPS). Second, while that will provide a level of end-to-end security, it will present a new challenge for the functions and services that rely on inspecting the content of Web traffic. Some of these services and functions will concern security, such as auditing and access control. The challenge comes from two directions: (1) the standard Web proxies of today pass the HTTPS traffic through, and (2) Web proxies are somewhat global (aggregating Web traffic from many users or applications) and agnostic to personalization to an individual user's or an application's context and requirement. We developed a *personal proxy* that is capable of handling both HTTP and HTTPS traffic, and demonstrated its use in tackling the threat of phishing attacks. This personal proxy will be a useful tool for implementing functions and services that require inspection of Web traffic content.

Introduction

The ability to intercept normal interaction between application components enabled a number of useful functions such as monitoring and auditing, adaptive failover, load balancing, and (last but not least) enforcement of security policies. Obviously, the need for many of these functions is already felt in the context of Web-based applications. The use of Web proxies by organizations to monitor and protect Web-based applications running within their networks, the use of load balancing mechanisms in server farms, and handling cross-domain exchanges are cases in point.

A number of interception-based functions require *deep inspection* of the traffic, meaning operations that need to access

the content of the payload, not just the HTTP header information. Web proxies can do this job perfectly for HTTP traffic, but not for HTTPS traffic. The reason is that HTTPS is the secure version of the HTTP protocol, and HTTPS payloads are encrypted by Transport Layer Security and are not meant to be inspected or modified by interlopers like the proxy.

As important services increasingly become Web-enabled and as the task of setting up HTTPS becomes routine, we

“We developed a personal proxy that is capable of handling both HTTP and HTTPS traffic, and demonstrated its use in tackling the threat of phishing attacks.”

expect that increasing Web traffic will move over to HTTPS to provide a level of security that the users have come to expect (e.g., the padlock sign on the browser). This gain in one aspect of security (i.e., site authentication and defense against confidentiality and integrity attacks on the information during the transit) makes it difficult for functions that require access to the content, such as auditing and monitoring, application level rate limiting, application level adaptive caching, context-specific failover and load balancing, and so forth. In addition, as Web services become the de-facto mechanism of information exchange, proxies are likely to play a key role in handling cross-domain issues. For example, opening HTTP connection to Web sites other than the one from which the current Web page was served is usually not permitted from the browser,

but the application may need to interact with services from other Web sites. Using a proxy is one solution that is often used to get around that problem. The problem gets more complicated if different services are at different security levels. If that transaction happens over HTTPS, the standard proxies will be of no use – one must use a proxy like ours that can proxy HTTPS.

The global and impersonal nature of the proxies poses another challenge. Unlike a firewall (that deals with many protocols including HTTP and many ports including those used by Web services), a Web proxy is narrowly focused on the HTTP traffic. However, like a firewall, a Web proxy covers multiple hosts, users, and applications in an aggregate form. The wide variety of Web applications and their range of importance and sensitivity – from financial transactions like banking and shopping to social interactions over Facebook, Web-based e-mail, and chat – will demand an unforeseen level of personalization or application-specificity in monitoring, auditing, access control, rate limiting, or load balancing solutions. We claim that the aggregate and one-size-fits-all nature of Web proxies will make the Proxy-based Solutions situated at the Internet Service Provider (ISP) or at corporate boundaries insufficient and less acceptable.

On one hand, the users will be less comfortable disclosing their personal preferences and requirements to the remote proxy that they do not own and control themselves. While understanding and enforcement of the policy may be a daunting task for some users, they will still demand canned policies that they can turn on. Think of setting your browser's security settings, but different settings for Facebook and your bank, and even different settings for different Facebook users in your household that you can control. Then, there will always be a group of technology-literate users questioning the adequacy of protection of personal data

and the quality of enforcement offered at the remote proxy. On the other hand, because the remote proxy aggregates traffic flow from multiple users and applications, they are ill-equipped to enforce policies and preferences that are highly specialized (personalized) for individual applications or users without mutual interference.

We argue that we need a *personal Web proxy* that will do the following:

- Be situated near the user or the application it proxies (it is even possible to have dedicated proxies for each application), and is controlled by the user or the owner of the application it is proxying.
- Enforce the user's or the application's policies and personal preferences that can be easily plugged in.
- Be able to inspect HTTPS traffic without compromising the security gains contributed by the HTTPS protocol.

The envisioned personal proxy is analogous to personal firewalls: As personal firewalls bring firewall capability near to the user's host from the network edge, the personal proxy will also push proxying capability from the network edge closer to the user or the application. Furthermore, the personal proxy is a valid application level proxying mechanism that can be easily customized for the application or user at hand; it provides an easy way to introduce additional application or user-specific functionality in the HTTP/HTTPS path.

There are a number of software engineering reasons supporting the need of a separate proxy, as opposed to embedding the needed additional functions into the application components it mediates between. First, the proxy adds a separate layer of protection (another process to corrupt – a crumple zone, if you will), and provides stronger isolation guarantees (defense against memory corruption attacks) and increased flexibility. The proxy is less complex than the browser that has to support applications ranging from streaming media to Java applet, and provides a smaller attack surface. Since the proxy is a dedicated process, it can be protected using technologies that implement process protection domains, such as SELinux [1] or Cisco Security Agent [2]. Second, a personal proxy offers a good middle ground between the two extremes, dealing with the aggregate of interactions at the network edge or modifying each application. A browser plugin-based implementation will not be able to control or monitor non-browser applications that may use HTTP or HTTPS and should be

subject to the same user-defined policies and preferences. To cover this situation, one either assumes (somewhat unrealistically) that all applications interacting over HTTP/HTTPS use the browser or are forced to develop similar embedded capabilities for each of those non-browser applications. Furthermore, the corporate or ISP proxy may not be able to enforce policies of individual applications and users at the network edge. It is easier to implement user- or application-specific policies and behavior into a personal proxy that runs on the user's host and, using firewall rules, mandate that the only way HTTP/HTTPS traffic gets in or out is through the proxy. Third, any mechanism that enables flexible and customizable introduction of additional behavior, constraint enforcement, and monitoring without requiring costly (and sometimes

“... a personalized proxy can be used to protect the user from divulging personal information to malicious Web sites ...”

impossible) code changes in the original application is a valuable software engineering asset. The personal proxy performs this job adequately. Other than ensuring that the HTTP/HTTPS traffic flows through it, no code change is necessary for the applications that interact through it. Finally, to be general and to support all kinds of monitoring and inspection use-cases, the additional user- or application-specific policies and behavior must be inserted before traffic is encrypted with the remote site's key. To illustrate the point, note that Chinese users are able to bypass governmental scrutiny enforced at their network edge by interacting with encrypting proxies outside China. While our proposed personal proxies are controlled by the user/application it covers (as opposed to any government agency), there are use-cases (e.g., parental control, cross-domain security policy enforcement) where personal proxies provide a better solution than the browser-embedded checks or proxies at the edge.

Under Department of Homeland Security (DHS) funding, we have developed a customizable Web proxy that handles both HTTP and HTTPS protocols. For HTTPS, the proxy works by establishing two System Specification Language

(SSL) connections: one between the browser and the proxy, and the other between the proxy and the remote Web site. The customization happens by configuring the proxy's chain of interceptors. The proxy can be placed near the user, on the user's computer, or at the user's home router box. We have demonstrated how such a personalized proxy can be used to protect the user from divulging personal information to malicious Web sites (i.e., defense against phishing attacks). We have started investigating other uses of the proxy, such as auditing inter-agent communication in a semantic Web application so that the recorded interactions can be used by machine-learning algorithms that aim to learn and improve how the agents achieve their tasks. In this article, we briefly describe the architecture and operation of this personal proxy; a detailed description and the anti-phishing application appears in [3].

Architecture of the Personal Proxy

Figure 1 illustrates the design of the personal proxy, which consists of four main modules that are implemented on top of Jetty, a popular open-source Web server written in Java [4]. The *plugin framework* provides a means for integration of custom reactive and proactive behavior. In the first application of this proxy, all anti-phishing checks were implemented as a set of plugins for this module. A plugin can be one of the following three types, depending on its role in the overall control flow and threading logic:

- **Data plugins.** Each data plugin is invoked on every request and associated response. A data plugin is used for handling the header and payload data based on a specified security policy. For example, a proxy could be configured to record all or selected parts of Web traffic as part of a parental control policy. Recordings can be persisted securely on the disk.
- **Checks.** These plugins are organized in a chain, and intercepted requests flow through these checks like a pipeline. An individual check exits with either a *break* or a *continue*. A *continue* indicates that the request goes to the next stage, possibly with some additional metadata tagged to it. *Breaks* can be of two kinds: A negative break indicates that the request is to be blocked, while a positive break indicates that the request is to be accepted. In either case, a break implies that the rest of the pipeline stages are not executed.

This semantics of checks is amenable to modular implementation and integration of security policies.

- **Probes.** In contrast to checks and data plugins, which only execute reactively when triggered by requests or responses, probes allow us to embed proactive behavior into the proxy. Probes contain dedicated threads that trigger monitoring functions at regular configurable intervals. The probes can be configured to visit specified URLs and scheduled intervals to collect data that is relevant for the security policy context. For example, in the case of defending against phishing attacks, the probes were used to check for changes in an Internet Protocol address or security credential of the banks or financial sites registered by the user.

The lower part of Figure 1 displays the remaining three modules. The modules act as access paths into the proxy. The *HTTP Proxy* listens on a configurable network port (e.g., 8080) for incoming HTTP requests, and dispatches the requests to a main handler (*InterceptHandler*), which in turn makes strategic use of the plugins. This flow is similar in the case of the *HTTPS Proxy*, except that it listens on a different network port (8443) and uses a custom extension of the *InterceptHandler* (called *SslProxyHandler*) that intercepts HTTPS connect requests and facilitates subsequent interception of all HTTPS requests in that session. The third access path, HTTPS Requests, is for management of the proxy through an administration console. Management functions include changing the order of plugins and their respective importance weights as well as customization of user-specific data. The administrative interface is optional for out-of-the-box deployment, where the proxy is preconfigured and preloaded with appropriate plugins that enforce the desired policy. We do not anticipate that the internal details are important for most of the users (beyond pointing their applications or browsers to the proxy). The users who write and package custom policies for different users and applications will need to know the details of plugins. A better policy interface, supporting a generation of plugins (which can be added to the proxy by editing a configuration file) from higher-level policy specification, and a better way to inspect the policies encoded in existing plugins, is part of our future work. Once this policy interface is in place, these users will also be shielded from the internal details and complexities of the plugin architecture. If the internal details change because of evolution of the

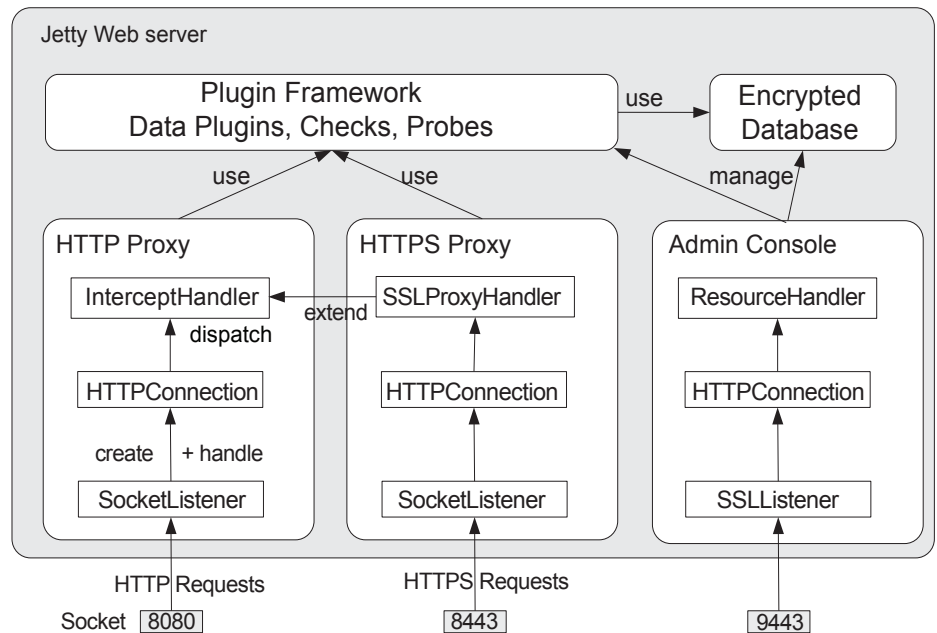


Figure 1: *Functional Architecture of the Personal Proxy*

Jetty code base/Web services specification, only the policy interface implementation will need to change.

Placement Options

The standard deployment of the proxy is on the end user's computer. Although this puts a small load on the Central Processing Unit (CPU), memory, and disk resources on the end system, it has the benefit of putting the proxy under direct control of the end-user. Our understanding is that end-users feel uncomfortable with disclosing personal and sensitive information (preferences, policies) to external parties, but are more amenable to providing this information to local components as long as it doesn't leave their machine. Since many end-users own either a wireless or DSL router and since these devices already ship with Web server capabilities, we investigated deploying the proxy on a Linksys WRT54G wireless router running OpenWrt [5]. Another option is to run the proxy on a home router, which has the benefits of increased security through stronger isolation from a potentially virus-infected desktop, and a new value-add for router manufacturers. On the downside, the very limited CPU and memory resources of the home routers, especially wireless routers, significantly lowers the performance of the proxy.

Insertion Into HTTP(S) Flow

Insertion of the proxy into the non-encrypted HTTP client-server path is straightforward and involves changing the client application's proxy settings (e.g.,

HTTP Web browser). To prevent an attacker from replacing the proxy setting to a proxy of his own, and to ensure that any application using HTTP/HTTPS is subject to the security policy enforced by the personal proxy, firewall rules should be set to only allow outgoing Web traffic through the personal proxy. For intercepting encrypted requests from client application that uses HTTPS, the client application's (such as the browser's) proxy settings are changed accordingly to redirect requests to personal proxy's HTTPS port. However, describing how appropriate security associations are established is slightly more involved (see Figure 2, next page).

In a regular use case without any HTTPS proxy, SSL relies on a Public Key Infrastructure for connection establishment [6]. Following a general description of the SSL protocol, the client issues a connection request to the server, which the server acknowledges with a response containing a certificate signed by a certification authority (CA). The client then continues to perform a set of checks on the server certificate, the main one of which is to verify that the CA's signature is valid. In most cases, SSL transactions essentially establish a unidirectional trust relationship between the browser and the target Web server via a commonly trusted CA.

With the proxy in the mix, the protocol becomes a little more complex. The proxy takes on the role of a server when communicating with the browser and the role of a browser when communicating with the target Web server. This requires the proxy to dynamically generate X509 certificates for each Domain Name

COMING EVENTS

October 14-16

Software Assurance Forum
Gaithersburg, MD

<https://buildsecurityin.us-cert.gov/daisy/bsi/events/930-BSI.html>

October 19-23

*International Conference on
Object-Oriented Programming Systems,
Languages, and Applications*
Nashville, TN

www.oopsla.org/oopsla2008

October 20-21

*National Defense Industrial Association
Technical Information Division Conference*
Huntsville, AL

www.ndia.org/meetings/9010

October 26-30

SIGAda with SAMATE
Portland, OR

www.sigada.org/conf/sigada2008

November 10-14

Agile Development Practices 2008
Orlando, FL

www.sqe.com/agiledevpractices

November 11-14

*19th IEEE International Symposium on
Software Reliability Engineering*
Seattle, WA

www.csc2.ncsu.edu/conferences/issre/2008

December 8-12

*Annual Computer Security Applications
Conference*
Anaheim, CA
www.acsac.org

April 20-23, 2009



*21st Annual Systems and Software
Technology Conference*
Salt Lake City, UT

www.sstc-online.org

COMING EVENTS: Please submit coming events that are of interest to our readers at least 90 days before registration. E-mail announcements to: nicole.kentta@hill.af.mil.

System name it is proxying² certified by its own CA³ (called PB CA in Figure 2). During installation, the Web browser's (and any other application's using HTTPS) settings are configured to trust signatures from the PB CA. As a result, the overall trust relationship between browser and target Web server can now be decomposed into two daisy-chained relationships, one between the browser to the personal proxy, and a second between the personal proxy and the target Web server.

Does the proxy introduce additional security vulnerabilities by breaking the end-to-end encryption between browser and Web server? The answer to this question depends on the relative trustworthiness of the proxy compared to the browser and target Web server and where it is deployed. Consider the case where the user does not use a personal proxy, but thinks that his desktop and the servers he uses are more secure than the ISP server through which he uses the Internet. The ISP server may co-host other applications, and if it does not have the latest security patches installed, such a setup would significantly lower the overall security of Web transactions flowing through it. On the other hand, if the personal proxy is co-located with the Web browser on the same desktop, we would expect it would be more difficult for attackers to subvert or corrupt the Java-based stand-alone proxy process (which only listens on localhost) compared to a C++ Web browser running Javascript. In both cases, data is never sent unencrypted over the network, so the guarantees provided by SSL across host boundaries are not affected.

Performance Overhead

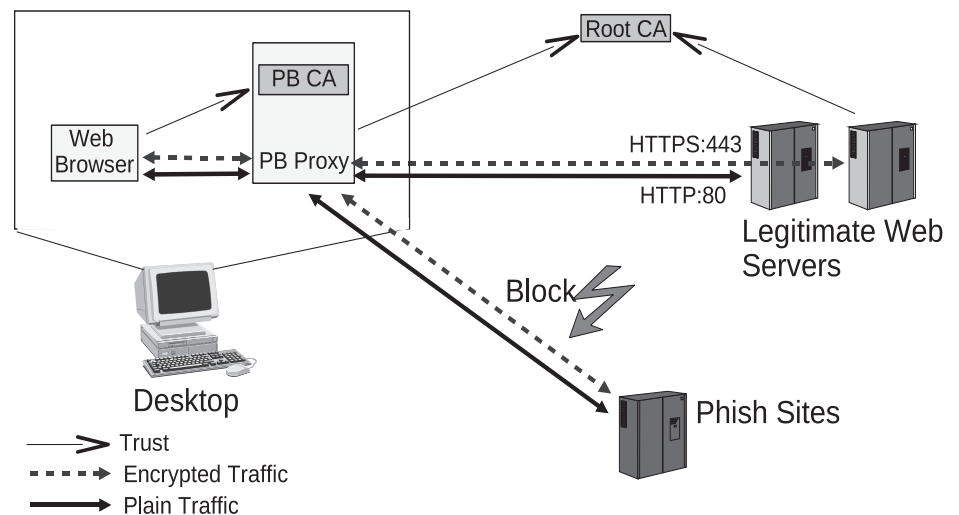
Introduction of a clearly noticeable delay presents increased resistance to adoption

of the new technology. To minimize performance impact, we implemented the proxy on top of the high performance Jetty Web server and implemented various optimizations in the SSL proxy architecture to keep request latencies (i.e., elapsed time between a request and its response) within user acceptable levels. In this section, we use *overhead* to mean the increase in request latency due to interception of HTTPS traffic by the proxy.

We measured the overhead in a lab setting by visiting HTTPS sites without the proxy and with the proxy configured with a number of anti-phishing checks. The mean time to load the visited pages through the proxy was twice the mean time to load the same pages without the proxy (excluding any user interaction like typing a password for both cases). However, the variance of load time was comparable to the mean (not surprising because we were visiting sites on the Internet), and even an overhead of roughly 100 percent was not distinguishable from the noise (as noted by external field testers, the delay introduced by the proxy does not noticeably impact the user Web surfing experience). Much of this overhead can be attributed to crypto operations and session multiplexing performed in Java. We expect the plumbing overhead to stay independent of the policy checks enforced by the proxy.

We also compared the round-trip latencies between an auditing configuration (when the proxy is simply recording) and a policy enforcing configuration (loaded with anti-phishing checks). We found that the two distributions are not significantly different as their inter-quartile ranges overlap to a large extent (from 200 to 1,500 milliseconds [ms]) and both distributions have a large number of outliers (some even greater than

Figure 2: Personal Proxy as a Trusted Middleman



50,000 ms). We suspect that available network bandwidth to the external Web sites together with available CPU resources of those sites have the biggest impact on round trip latencies, which is why the distributions looked similar.

Related Work

Various HTTP and HTTPS proxy implementations exist for debugging purposes (Burp proxy [7], Charles proxy [8]) and Web filtering (WebCleaner [9], Privoxy [10]). There are also a number of commercial network layer tools (e.g., eSafe's Web Threat Analyzer [11], McAfee IntruShield [12]) that can inspect Web traffic, including HTTPS that work at the enterprise layer. In many cases, these are geared for regulatory and auditing compliance, the DHS-funded research focused on transparent inspection of SSL traffic exclusively for regulatory purpose. However, we were unable to find a proxy that could be used as a general purpose middleware construct for customized user and application-specific policies.

Conclusion

We have been developing advanced middleware technologies that enable adaptive behavior, quality of service (QoS) management and QoS-based adaptive behavior in distributed systems over the past several years [13]. In doing so, we have developed middleware constructs for handling different styles of distributed interaction (e.g., distributed objects, publish-subscribe, group communication) over a number of protocols (e.g., socket-based, Common Object Request Broker Architecture or Remote Method Invocation). The present work involving HTTP and HTTPS interception complements that line of successful work, and enables us to introduce advanced middleware capability to distributed systems that use these protocols. The concept of a personal proxy has the potential to fill an important and emerging gap in the current Web-based systems architecture.

However, as noted earlier, the personal proxy is still in its early stages – we only have a prototype implementation that is demonstrated with anti-phishing checks, and have just begun exploring its use in other contexts.

A number of software engineering and usability issues also need additional work, including an easy way to inspect enforced policies and the ability to define policies at a higher level of abstraction that can be automatically translated into executable code that can be integrated into the plug-

in framework. These are the next steps we hope to tackle. ♦

References

1. Loscocco, P., and S. Smalley. Integrating Flexible Support for Security Policies Into the Linux Operating System. Proc. of 2001 USENIX Annual Technical Conf. USENIX Association, Berkeley, CA: 2001.
2. Cisco. "Cisco Security Agent-Enterprise Solution for Protection Against Spyware and Adware." Cisco White Paper <www.cisco.com/en/US/prod/collateral/vpndev/ps5707/ps5057/prod_white_paper0900aecd8020f438.html>.
3. Zodgekar, Sameer A. Identity Theft: A High-Tech Menace. ICFAI University Press. Apr. 2008.
4. Mortbay.com. The Jetty Java Web Server Vers. 5.1. 2007 <www.mortbay.org/jetty-6/>.
5. Openwrt.org. The Linux Distribution for Wireless Freedom <http://openwrt.org>.
6. Wagner, D., and B. Schneier. Analysis of the SSL 3.0 Protocol. Proc. of the Second USENIX Workshop on Electronic Commerce. Oakland, CA: 1996.
7. Portswigger.net. The Burp Proxy Tool Vers. 1.1. 2008 <www.portswigger.net/proxy>.

net/proxy>.

8. Charles Web Debugging Proxy. About Charles. <www.charlesproxy.com>.
9. Kuhnast, Charly. "Junk Zapper." Linux Magazine June 2004 <www.linux-magazine.com/issue/43/Charly_Column.pdf>.
10. The Privoxy Team. Privoxy 3.0.8 User Manual. 2008 <www.privoxy.org/user-manual/index.html>.
11. Aladdin.com. "The eSafe Web Threat Analyzer Audit." 2008 <www.aladdin.com/esafe/solutions/wta>.
12. McAfee.com. "McAfee Network Security Platform Data Sheet." 2007 <www.mcafee.com/us/local_content/datasheets/ds_network_security_platform.pdf>.
13. BBN Technologies. The QuO Group at BBN. "Distributed Systems Technology Group Papers." <www.dist-systems.bbn.com/papers/>.

Notes

1. This work was supported by the DHS Advanced Research Projects Agency under contract number NBCHCO50096.
2. To increase generation performance, key pairs can be reused across certificates.
3. Alternatively, the PB CA can be signed by a common root CA.

About the Authors



Partha Pal, Ph.D., is a division scientist at BBN Technologies' National Intelligence Research and Application business unit. His research interests include adaptive and survivable distributed systems and applications, and technologies that enable adaptive behavior and survivability. Pal has published more than 35 technical papers in peer-reviewed journals and conferences and is a senior member of the Institute of Electrical and Electronic Engineers (IEEE) and a member of the Association for Computing Machinery. He received his master's and doctorate degrees from Rutgers University, New Brunswick, NJ.

BBN Technologies
10 Moulton ST
Cambridge, MA 02138
Phone: (617) 873-2056
Fax: (617) 873-4328
E-mail: ppal@bbn.com



Michael Atighetchi is a scientist at BBN Technologies' National Intelligence Research and Application business unit. His research interests include security and survivability, intelligent agents, and middleware technologies. Atighetchi has published more than 20 technical papers in peer-reviewed journals and conferences, and is a member of the IEEE. He holds a master's degree in computer science from University of Massachusetts at Amherst, and a master's degree in information technology from the University of Stuttgart, Germany.

BBN Technologies
10 Moulton ST
Cambridge, MA 02138
Phone: (617) 873-1679
Fax: (617) 873-4328
E-mail: matighet@bbn.com