Transparent Insertion of Custom Logic in HTTP(S) Streams Using PbProxy

Cost and testing considerations limit the acceptance and deployment of technologies that make information exchanges more secure, reliable, semantically understandable, and self-improving. PbProxy is a flexible proxy that enables transparent insertion of custom logic into HTTP and HTTPS interactions. It has successfully been used to facilitate behavior-based prevention of phishing attacks, machine learning of Web service procedures, and Web browsing over disruption-tolerant networks by injecting custom logic into existing applications and communication streams. PbProxy encapsulates common functionality into a proxy base and supports customizable plugins to foster code reuse.

Developers have used HTTP since the World Wide Web’s beginning as an application-level protocol for distributed, collaborative, hypermedia information systems – that is, systems that include various media types (such as graphics, audio, video, and plaintext) linked together to form a nonlinear information medium. Early usage was predominantly centered on webpage retrieval, but HTTP is now a foundational building block for many current and emerging software technologies, including service-oriented architecture (SOA) and Web services.

Numerous current and emerging applications hosted in the network cloud routinely use HTTP for bulk data transfers and remote interactions. The HTTP specification, combined with Representational State Transfer (REST), makes intermediaries an important part of the overall system architecture, along with stateless and self-describing requests. HTTP admits several types and multiple layers of intermediation between the client and server, including proxies, gateways, and tunnels. However, end-to-end encryption in HTTP Secure (HTTPS) – which combines HTTP with Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols – limits the intermediaries’ capability. Many of today’s distributed applications use HTTPS and could greatly benefit from recent research that takes advantage of the protection offered by HTTPS and overcomes

Michael Atighetchi, Brett Benyo, Abhimanyu Gosain, Robert MacIntyre, Partha Pal, Virginia Travers, and John Zinky
Raytheon BBN Technologies
the limitations imposed on the intermediary to make communication and information exchanges safer, reliable, semantically understandable, and self-improving over time. However, if using the new technologies requires changing existing client or Web service code, the cost associated with technology adoption would be prohibitive in many deployments.

The PhishBouncer Proxy (PbProxy) addresses this problem by leveraging provisions supported by HTTP and HTTPS to transparently insert customizable control and observability. Here, we describe the use of PbProxy in several different research areas, including advanced information assurance, machine learning, computer-assisted training, and disruption-tolerant networks. The common theme underlying these efforts is the fact that they all involve HTTP- and HTTPS-based distributed applications and must insert customized behavior into the HTTP and HTTPS interactions to achieve respective research objectives. Using PbProxy on these different projects demonstrates the usefulness of its plugin architecture, which lets developers easily add custom logic for handling requests and responses. A detailed description of PbProxy’s plugin architecture, together with protocol details on transparent HTTPS interception, is available elsewhere. PbProxy is available as open source software under a modified BSD license at www.phishbouncer.com.

Behavior-Based Prevention of Phishing Attacks

Most existing anti-phishing products follow a signature-based approach in which users, ISPs, and security staff at financial companies provide suspected URLs as input to a centralized blacklist service, which disseminates vetted blacklists to end clients (mostly browser extensions) for enforcement. This approach ties the anti-phishing products’ effectiveness to the accuracy and timeliness of signature updates. Therefore, attackers always have a window of opportunity during which a significant fraction of end clients operate without updated signature protection.

The PhishBouncer effort’s goal is to implement a behavior-based approach to detect and prevent phishing attacks. This approach relies on a strategic combination of attribute-based checks to implement both reactive and proactive anti-phishing defenses; in demonstrations, it has protected against previously unknown phishing attacks.

PbProxy forms an interception substrate, on top of which new functionality can be easily implemented and combined with existing checks through a flexible plugin architecture. Although the complete plugins set is beyond the scope of this article, following are a few example security checks that can be injected into HTTP(S) streams through PbProxy plugins.

**Filtering HTTP Requests**

The URL suspicious check tries to identify phishing sites by looking at their URL characteristics (puny-encoding and edit distance).

Similarly, the leaking user data check searches through outgoing traffic for sensitive user information, such as social security numbers, and the phish signature match check, which compares URLs against a phishing signature feed. We’ve observed correct interception semantics for the HTTP GET, HEAD, PUT, and POST methods.

**Filtering HTTP Responses**

The HTML crosslink check looks at responses from nonregistered sites and counts the page links to any registered site (in particular, those with image cross-links). Having many cross-links is indicative of a phishing site.

**Control Interception through Micro Protocols**

The key idea behind the false info feeder check is that a phishing site is likely to indiscriminately accept user-supplied data because it has no way of knowing whether such data is valid or not. We can use this fact to detect the presence of phishing sites.

For example, PhishBouncer might decide to send a bogus user name and password combination (via HTTP POST) to a site that previous checks have declared suspicious. If that site doesn’t reply with an error, it further increases the site’s “phishyness” factor.
HTTP Request and Response Correlation

The suspected by ImageAttribution check and its associated data plugins look for image similarities among sites the user has previously registered with PhishBouncer and sites the user is trying to visit. If the images are similar but the domains are different, it indicates that the user is trying to go to a phishing site that either displays back a local copy of the image or links to the real image.

The referred by Webmail check looks at HTTP requests' referrer headers and declares requests suspect (with low confidence) if they originate from a Web-based email provider.

Inline Calls to Third-Party Services

The idea behind the domain too young check is to flag sites that are hosted on very young DNS domains, as phishers frequently create mirror domains such as bancofbar1.com (rather than bankofbar.com) and move on to new domains (bancofbar11.com) as ISPs take down active ones.

Acting on Multiple Checks Results

If an HTTP request emerges from the chain of checks with a vector of check results, it will enter the aggregation plugin, which uses a weight-based threshold scheme to make the final determination as to whether the request is safe.

Experimental data shows that HTTPS proxying introduces an overhead of roughly 100 percent; we contribute much of this overhead to crypto operations. Overhead distributions between instances with all checks enabled and all checks disabled showed overlapping interquartile ranges from 200 to 1,500 milliseconds (ms) per request.

Machine Learning of Web Service Procedures

The goal of the Poirot project (for plan order induction by reasoning from one trial) is to design an architecture and set of learning components that collectively form a multistrategy learning system for Web service procedures. The project’s primary innovation is in exploring techniques for applying various learning and reasoning strategies to learn hierarchical procedures from one example (or several examples at most). Poirot’s objective is to form a generalized hierarchical task model from “observed” semantic traces of Web service transactions sequences.

Poirot uses PbProxy to:

- create semantic traces from raw observations,
- support mixed human-computer interactions that let learning components ask for human input if they get stuck, and
- overlay learned workflow information on Web applications for online training purposes.

Semantic Workflow Capture

Before learning can occur, we must capture a trace of user interactions. For Poirot, this requires converting the raw HTTP traffic into a ServiceCall, which is a semantically annotated object defining the user’s action, its inputs, and the resulting changes to the displayed page.

The ServiceCall Recorder, a PbProxy data plugin, is responsible for this task and consists of three subcomponents. The ServiceCallExtractor constructs service calls by waiting for a matching response to pair with a request, ignoring included files and images and Ajax calls. Redirects are handled by returning only the final response page that the user will see in the browser.

Once a ServiceCall has been created, we pass it on to the ServiceCallInterpreter, which determines the action the user took to initiate the request and what changed on the resulting page. Hyperlink actions are straightforward because they result in an HTTP GET request, but ASP.NET postbacks are more complicated. For a postback, the state of every control on the page is transmitted to the server in the request. The interpreter scans the request parameters, comparing their state to the values from the previous call to determine the action taken and what input parameters changed. The interpreter uses an HTML parser to perform a tag-by-tag diff to see what changed on the page; it then sends the results to the third component, the SemanticTranslator.

The SemanticTranslator uses definitions of the possible actions and Web controls defined in Poirot’s Learnable Task Modeling Language (LTML), acquired by parsing the raw
HTML pages. The translator converts the data in the ServiceCall object to LTML. Through the PbProxy Admin servlet, users can instruct the translator when to output the stored trace of a file, and when to start and stop recording.

In addition to annotating HTTP requests, Poirot also supports annotations on GUI actions that don’t directly lead to observable network traffic, such as entering text in a text field or selecting a radio button option. The EditTracker plugin enables transparent instrumentation of web pages to capture those annotations by inserting JavaScript into the onChange attribute of these controls to store additional data on these events, such as timestamps. When the next postback does occur, these timestamps are encoded in a hidden tracking element. When the ServiceCallInterpreter gets the ServiceCall with these tracking elements, it splits up the ServiceCall into multiple calls based on the timestamps of the tracked actions.

For example, suppose the user updates a text field, selects a radio button option, and finally presses a button. The tracking code will capture the timestamp of the text field update and radio button selection, passing them along with the button-press HTTP request. The interpreter will then create three service calls: one each for the text field set, radio button selection, and the button press.

The Poirot Integrated Learner uses the semantic trace to learn how to use a Web application so it can assist new users, offer technical support, and help identify usage patterns during data mining.

**Mixed Human-Computer Interactions for Web Server Access**

Figure 1 shows the WatcherPlugin, which mediates access to a Web server from two clients. This plugin permits a Poirot client and a user at a browser to share a single session with a Web server. The WatcherPlugin operates like an autopilot feature — when the human is in control, the Poirot client watches; when the Poirot client is in control, the human watches.

**Client/browser pull.** Supporting a monitoring usage pattern is difficult because an HTTP server, or proxy, can never push information to the monitoring client; the client must request information. If a client sends a request, and the proxy has no more cached replies to send back, the client must continue to make requests.

To ensure that the browser continually (but invisibly to the user) pings the proxy, we inserted JavaScript into pages returned to the browser. The JavaScript uses Ajax to periodically send XMLHttpRequests to the Web server; these were intercepted by the plugin, which acknowledged the requests until the next page was available.

**Switching control.** We controlled the switch between user mode and Poirot mode by generating special switch mode messages. For browser clients, the plugin inserts JavaScript to pop-up a page with a button for the user to return control to Poirot. When the user presses the button, a switch mode request is sent to the Web server and intercepted by the plugin to switch modes.

**Redirects.** The final piece of the control process involves the use of redirects. If at any point a client requests a page that’s not the next page in the proxy’s queue, then the client is sent a
redirect to that page. In particular, this is how the next page is served to a browser that is pinging the proxy, awaiting a page from the Web server.

Using a proxy as a mediator between two clients is useful in training and help-desk scenarios, or wherever inexperienced users could benefit from watching experts, or experts could benefit from watching problems encountered by less experienced users.

Self-Instructing Web Applications
The functionality implemented in the HelpPlugin (see Figure 1) modifies the original HTML of an existing Web application to add help widgets using standard HTML, Cascading Style Sheets (CSS), and JavaScript. The HelpPlugin uses JavaScript to track user actions through Ajax calls. It displays help messages and graphics in a dedicated frame at the top of the browser window and highlights portions of the original webpage in the rest of the window.

Encapsulating the help system into a PbProxy plugin lets us use the help system as a separate layer on top of an existing Web application, without directly changing any of the original webpages.

Figure 2a shows a screenshot of the self-instructing mobilization and deployment tracking information system (MDIS) application. The labels show the results of various actions implemented by the HelpPlugin. Figure 2b shows the interaction patterns associated with the HelpPlugin in more detail.

The plugin handles two kinds of requests:

- normal HTTP requests for content from the original Web application (original content), and
- help requests originating from the browser’s help system (help content).

Normal HTTP requests for page content are passed through to the underlying Web server. The HelpPlugin intercepts the returning original page and adds HTML help content and JavaScript to the original HTML, returning an altered page to the browser. The help content includes an internal frame to display help messages and graphics. The added JavaScript tracks user actions and updates the help displayed as users progress through a task.

For help requests, the tracking JavaScript reports user actions back to the help system via Ajax. The HelpPlugin intercepts the resulting help request and, using its dynamic task-tracking module, analyzes the user input and page state to determine which help pages to display next. It returns this information as a set of Actions encoded in an XML response, which the tracking JavaScript interprets to update the help display in the browser window as requested. The help content also includes custom stylesheets, graphics, and sounds. The HelpPlugin intercepts HTTP requests for this
content just like it handles the Ajax requests, responding directly to the request instead of passing it to the Web server.

Performance
After inserting PbProxy without any checks, the median round-trip times for Web services calls increased from 122 ms to 125 ms (with standard deviations of 8 and 3, respectively). Enabling all Poirot plugins led to a median value of 135 ms, with a standard deviation of 9 ms.

Browsing over Disruption-Tolerant Networks
Disruption-tolerant network technology allows delivery of information bundles even when there’s no connected path between end points. DTN can use unreliable and nontraditional links, such as data mules, interplanetary links, and transient radio links, as well as traditional IP subnetworks. DTN can be viewed as an overlay network that unifies a wide range of link types and lets pluggable components implement different routing and content storage algorithms. A typical DTN uses a bus as a data mule, which drives stored data to remote locations. As the bus arrives at each station, the bus’s DTN router uploads and downloads information bundles to the station’s DTN router. This router acts as the local nexus, connecting end users by any available link type, including IP, Wi-Fi, or even USB memory sticks.

To access Web content over DTN, the TCP-based HTTP protocol can’t be used because TCP depends on an end-to-end connection and webpage rendering requires multiple round-trips. However, Web browsing is possible if the HTTP traffic can be terminated in a local Web proxy and the user is told which content has been prefetched and which content must be fetched over DTN.

Figure 3 shows a prototype integration of DTN and PbProxy to insert the custom behavior described earlier into the generic HTTP and HTTPS interaction. The prototype’s main goal is to reduce the round-trips required to render a page and enable prefetched content reuse among multiple users at a remote site. PbProxy can function both as the client-side Web proxy to terminate the HTTP traffic and manage user feedback, and on the server side, as a Web crawler to prefetch content needed to render a webpage. With PbProxy integrated with a DTN, Web browsing could take one round-trip time to render a webpage and prefetch the page’s referenced links.

The prototype had two PbProxy plugins: one to handle the client side, and the other to handle the server side. The client-side plugin manages HTTP requests and user feedback. When an HTTP request arrives, the requested URL is checked against the local cache. The cache is designed to first check the local store and then use the DTN content-based access when the implementation is available. If the cache lookup times out, the HTTP request is bundled and sent to a server-side proxy. The client-side plugin also generates a status webpage, which informs users that the content is being fetched and offers options — such as an RSS feed — for user notification about content arrival. The status page can also display any feedback about the status of the end-to-end path, if the DTN router knows it. When the content arrives, the
client-side plugin pushes it to users through the particular scheme they selected.

The **server-side plugin** manages the fetching of content associated with the HTTP request. The URL is given to the HTMLUnit subsystem, which renders the base page and returns all of its HTTP references. It also fetches all references needed to render the page and crawls the embedded `href` links to a configured depth. Each URL's content is bundled and published into the local cache using the URL as the key. Breaking up the one HTTP request's content into multiple content bundles enables reuse by other requests and users. In the prototype implementation, each URL–content pair was sent back to the client side using a reliable delivery bundle. In the future, this pair will be sent using DTN content-based publishing.

HTTP over TCP has many characteristics that make browsing difficult over a long delay network, and some HTTP extensions for handling low-bandwidth connections actually hurt the situation. There are several HTTP challenges; here, we briefly describe three.

First, static content rendered on a webpage contains many URLs for embedded pictures and frames. Each embedded URL actually involves an additional HTTP request to fetch its content. For DTN-based Web browsing, all the URLs needed to render the page must be prefetched to minimize the number of round-trips. The server-side Web crawler could scan the HTML markup to extract the additional URLs. But, with embedded stylesheets and JavaScript, whole-page rendering must be emulated on the server side to find the generated URLs. PbProxy emulates a browser and extracts all resolved URLs.

Second, content requests are small and could potentially travel over a different networking transport. For example, content requests could be sent over a low-bandwidth connected network and the large content itself be returned over data mules. The client-side PbProxy can distinguish requests for the base page from embedded URL requests; base page requests can be sent with DTN priority fields and urgent delivery, taking an expedient path to the server-side proxy. Sending requests for embedded URLs isn’t necessary because those will be discovered when the page is rendered at the server side, thus saving precious connected bandwidth.

Third, cookies are sent by HTTP requests that can tell the Web server how to generate the webpage — querying a browser for its resolution, for example, and then changing content to fit. But if the content is to be shared, it must be generic. PbProxy can add or remove cookies to trick the server into generating different format versions for the same URL, but the DTN-PbProxy prototype didn’t investigate this feature.

Among the next steps for PbProxy are both architecture refinement to better support existing uses and uses in new domains. By using PbProxy on multiple projects, we learned that the initial control and dataflow architecture developed under PhishBouncer — consisting of checks, data plugins, probes, and an aggregator component — didn’t fit the DTN and Poirot use cases well. We were able to accommodate the new use cases through custom handling; however, we plan to change the plugin architecture’s data and control interfaces in future versions to better support new patterns for plugin composition.

Currently, we’re investigating advanced security and survivability methods for constructing intrusion-tolerant SOA-based systems. Plugins for transparent client and services protection through content inspection, rate limiting, voting, and failover are all necessary capabilities to ensure safe operation of SOA-based systems in both tactical and enterprise environments.

**Acknowledgments**

We acknowledge the support and collaboration of the US Department of Homeland Security (DHS) Science and Technology (S&T) Directorate, the US Defense Advanced Research Projects Agency (DARPA), and the US Air Force Research Laboratory (AFRL). PhishBouncer was sponsored by the DHS S&T Directorate under contract NBCHC050096. Spindle was supported by contract number FA8750-07-C-0169 under the DARPA Disruption-Tolerant Network Program. Poirot was supported by contract number FA8650-06-C-7606 under the DARPA Integrated Learning program. The views and opinions expressed here are those of the authors and not the sponsoring agencies.

**References**

Michael Atighetchi is a senior scientist at BBN Technologies’ Information and Knowledge Technologies business unit. His research interests include cross-domain information sharing, security and survivability architectures, and middleware technologies. Atighetchi has an MS in computer science from the University of Massachusetts at Amherst, and an MS in informatics from the University of Stuttgart, Germany. He is a senior member of IEEE and a member of the ACM. Contact him at matighet@bbn.com.

Abhimanyu Gosain is an engineer in BBN Technologies’ Mobile Networking Systems group. His research interests include developing routing protocols for low-energy ad hoc sensor networks and characterizing WLAN 802.11X protocol performance in outdoor environments. Gosain has an MEng in electrical engineering from Tufts University. Contact him at agosain@bbn.com.

Robert MacIntyre is a software engineer in BBN Technologies’ Information and Knowledge Technologies business unit. His research interests are in user interfaces in various domains, from computational linguistics to military logistics. MacIntyre has an MSE in computer and information science from the University of Pennsylvania. Contact him at rmacinty@bbn.com.

Partha Pal is a lead scientist at BBN Technologies. His research interests are in distributed systems and middleware technologies, focusing on survivability and security of distributed systems. Pal has a PhD in computer science from Rutgers University. He is a senior member of IEEE and a member of the ACM. Contact him at ppal@bbn.com.

Virginia Travers is a senior scientist in BBN Technologies’ Information and Knowledge Technologies business unit. Her research interests are in the design and implementation of user interfaces, and simulation-based training. Travers has a BS in computer science based training. Contact her at travers@bbn.com.

John Zinky is a lead scientist at BBN Technologies. His research interests include quality-of-service-adaptive systems at the network layer (disruption-tolerant networking), distributed object layer (QuO), and distributed agent layer (Cougaar). Zinky has a PhD in electrical engineering from Boston University. Contact him at jzinky@bbn.com.

Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.