Strong Authentication in Web Proxies

Domenico Rotiroti
ASPNET S.c.p.a.
via anagnina, 124
00046 Grottaferrata (Roma)
+39 06954514229
d.rotiroti@aspnet.roma.it

ABSTRACT
In this paper we present a way to integrate web proxies with smart card based authentication systems.

Categories and Subject Descriptors
K.6.5 [**Security and Protection**]: Authentication

General Terms
Security.

Keywords
Smart card, HTTP, Proxy.

1. INTRODUCTION
The HTTP protocol [1] provides two standard methods to authenticate client requests: "Basic" and "Digest" authentication [2]. While they are adequate in many practical cases, there are circumstances where they show some limitation, the most important one is that both of them are password based so the security level is not extremely high. Most internet applications use several strategies to enhance security that are not feasible for proxies: ssl to submit forms containing passwords, randomically generated session ids stored in cookies, etc. A natural evolution would be the introduction of strong cryptography in proxy authentication: smart cards and crypto tokens are rapidly diffusing in companies and governative agencies as an helpful tool for access control and user profiling.

2. GOALS AND MOTIVATIONS FOR THIS PROJECT
ASPNET is a government agency operating in the province of Rome (IT), having in charge the development of an eGovernment project for the cities in its area. The offices in the towns of this district are connected to a central data center through a set of dedicated high speed link, they use this private network to access all of our services. Each operator has been given a smart card to authenticate through our single sign on system and a set of "application roles" describing the actions that he is allowed to perform on ASPNET services and applications.

WWW 2006, May 23–26, 2006, Edinburgh, Scotland.
ACM 1-59593-323-9/06/0005.

2.1 Why strong authentication in web proxies?
Dealing with government agencies, authenticating and logging web traffic is a must. Moreover, having developed a smart card based single sign on system, it seems reasonable to use it with all of our services, even with web browsing.

3. IMPLEMENTATION
3.1 Prerequisites
Our main goal was to setup an authentication system plugging painlessly in ASPNET network: without the need for additional software on the client workstations, browser plugins or particular network modifications in the local offices. We wanted the operators to browse the web as they did before, only having to type the smart card's pin on the first page accessed. Building such a system and in the same time adhering to the standards (e.g. HTTP protocol) has been technically challenging, in the following subsection we will examine these challenges and how they have been solved.

3.2 Identifying users
As a starting point for our implementation we choose the well known squid proxy server, because it's feature-rich, stable and modular. Squid allows the developers to define new authentication methods [4] implementing a set of callbacks to be invoked by the proxy when various relevant events occur. So we began working on a new authentication schema, trying to limit the modifications in squid's code to a few places where assumptions done for the other methods were no more valid. The main difference between canonical authentication schemas and smart card authentication is browser's "complicity": when an unauthorized user requests a page, a proxy server configured for basic or digest authentication reply with a "407/Proxy Authentication Required" status code [1], the browser understands this request and asks the user to type username and password, which are then sent back to the proxy via an "Authorization" header. [fig1]

In a smart card based proxy authorization system, there's a new actor: a web server accepting https connections. The server must be reachable from the clients and must accept client certificates (in particular must accept certificates signed by the authority issuing our smart cards). We will refer to this site as *securesite* from now on. When the user begins browsing the web, this time the proxy will reply with a "307/Temporarily Redirect" status, and a "Location" response header pointing to *securesite*. The browser will send a new request and the user will be asked to type his smart card's pin. At this point certificate's data may be read

from a server side script running on *securesite*, and the user is finally identified. [fig2]

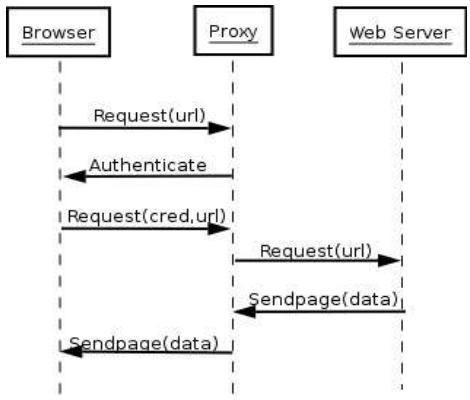

Figure 1. Request-Response flow in authenticated proxies

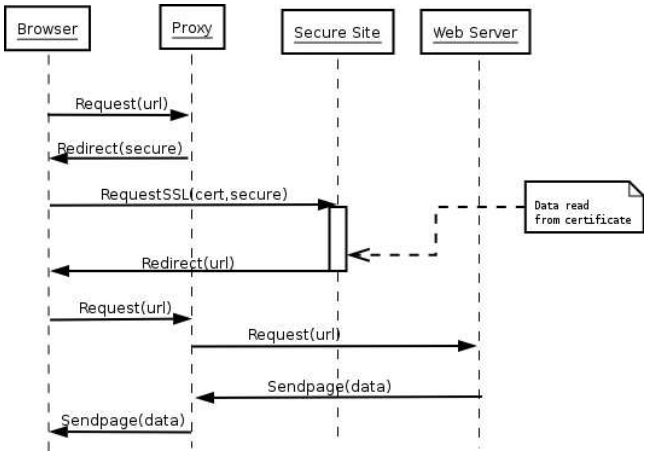

Figure 2. Smart-card authenticated proxy

3.3 Distinguishing authenticated requests

Since there is no "Authorization" header coming from the browser, we must find another way to block unauthenticated requests and satisfy the others. The approach we used is to try to keep proxy connections alive as long as possible after that they have been authenticated. This is similar to the work done for the NTLM support in squid [3], and is achieved linking the user data structure to the connection state data [4]. This solves our problem for all the requests going through the first connection, but leaves us with the problem of authenticating the other connections opened by the same client(e.g. to fetch images and subdocuments in parallel). Luckily, this is not an issue: the request will go through *securesite* but having not to type the pin again and with a small roundtrip the user will not notice the redirection and continue browsing as usual. Examining our logs, anyway, only a small subset of the page requests will be redirected.

Having a system capable to distinguish authenticated connections and to redirect the others to a trusted site opens the door to a new class of authentication types: many situations where the browser popup seems not to be enough could be easily handled with a custom server side script on a trusted site.

3.4 Dealing with https requests

When the client connects to a https site, all the traffic will go through a secure socket layer connection. In this case, the proxy only plays a marginal role: it will only send encrypted data back and forward, without knowing anything about headers and content travelling through it. The authentication routines are never called, so the connection is blocked.

To solve this problem we recurred to (semi-)transparent https proxying: the browsers are configured to connect to https sites directly and on our routers we use network address translation to forward these requests to the proxy. In this way our users always see a single server certificate and use their smart card for client authentication whatever site are they visiting, while it's up to the proxy to verify the real server validity. The request flow here is different from the non-https case: [fig3].

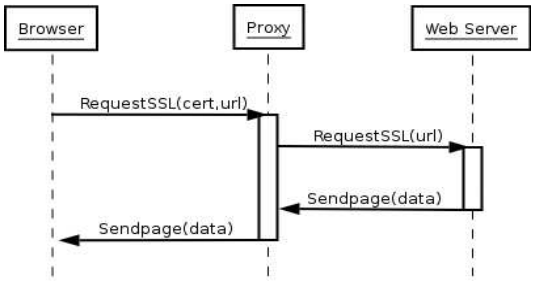

Figure 3. Transparent proxying of an https url

Never seeing the real server directly, clients cannot use any other certificate for client authentication purposes. While this situation does not arise so often, it is probably biggest limitation in our new authentication schema. At the moment, external sites requiring client authentication are handled by *ad-hoc* changes to the configuration.

4. RESULTS

In our first tests, the system shows good responsiveness and the feedback from the users is positive. To ease browser's configuration we deployed a proxy auto-configuration script [5], but still some setting must be changed by hand.

Further work could be done to improve compatibility with all sites/user-agents and to reduce network authentication traffic.

5. REFERENCES

[1] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol, HTTP/1.1", RFC 2616, June 1999.

[2] RFC 2617 HTTP Authentication: Basic and Digest Access Authentication.

[3] Chemolli F., Doran A., "Client-Squid NTLM authentication protocol description", http://squid.sourceforge.net/ntlm/client_proxy_protocol.html, 2003.

[4] Squid Programmers Guide, http://www.squid-cache.org/Doc/Prog-Guide/, 2004.

[5] "Navigator Proxy Auto-Config File Format", http://wp.netscape.com/eng/mozilla/2.0/relnotes/demo/proxy-live.html, 1996